PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 15/76

A1

(11) International Publication Number: **WO 99/52040**

(43) International Publication Date: 14 October 1999 (14.10.99)

(21) International Application Number: PCT/US99/07771

(22) International Filing Date: 8 April 1999 (08.04.99)

(30) Priority Data:
60/081,266 8 April 1998 (08.04.98) US

(63) Related by Continuation (CON) or Continuation-in-Part (CIP) to Earlier Application
US 60/081,266 (CON)
Filed on 8 April 1998 (08.04.98)

(71) Applicant *(for all designated States except US)*: STELLAR TECHNOLOGIES, LTD. [US/US]; Suite C–260, 849 Almar Avenue, Santa Cruz, CA 95060 (US).

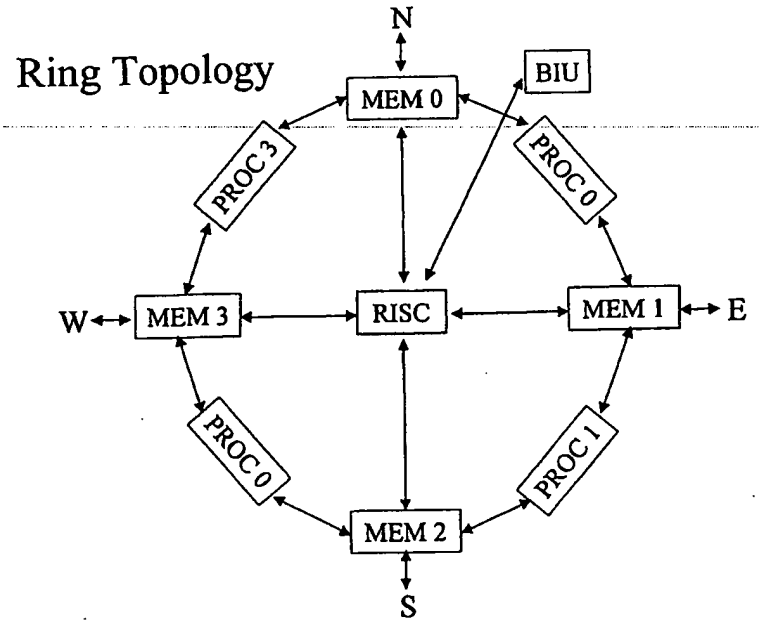
(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: RUBINSTEIN, Richard [–/US]; Suite C–260, 849 Almar Avenue, Santa Cruz, CA 95060 (US).

(74) Agent: STOLOWITZ, Micah, D.; Stoel Rives LLP, Suite 2600, 900 S.W. Fifth Avenue, Portland, OR 97204–1268 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: ARCHITECTURE FOR GRAPHICS PROCESSING



(57) Abstract

An embedded DRAM architecture specially adapted for graphics processing includes multiple processor engines and memory blocks arranged to form a ring topology. The processor engines include a standard execution unit and specialized execution units coupled to reconfigurable memory blocks to support MIMD style multiprocessing. Additionally, extensions to the instruction set architecture (ISA) are defined to dramatically improve performance in MPEG–2 and MPEG–2 encoding and other DSP applications.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

ARCHITECTURE FOR GRAPHICS PROCESSING

Technical Field

The present invention is generally in the field of digital computer architectures and, more specifically, is directed to circuits, systems and methodologies for digital signal processing utilizing shared, reconfigurable memory elements.

Background of the Invention

General purpose microprocessor cores are known for implementation into integrated circuits for a wide variety of applications. Digital Signal Processing (DSP) cores also are known for carrying out digital signal processing tasks. DSP cores are specially configured to efficiently process DSP algorithms. One example of a known DSP chip is the DSP 56002 processor, commercially available from Motorola. In order to achieve improved performance in DSP-related processing, the conventional approach is to combine a general purpose processor core together with a DSP core. The general purpose processor carries out Input/Output (I/O) tasks, logic functions, address generation, etc. This is a workable but costly solution. Additionally, evolving new applications require increasing amounts of memory and the use of multiple conventional digital signal processors. Additionally, power dissipation becomes a limiting factor in hardware of this type. The challenge, therefore, is to provide for improvements in digital signal processing performance while containing or reducing costs.

In view of the foregoing, one object of the present invention is to provide an improved computer architecture that utilizes available memory more efficiently in DSP systems. Another object is to reduce the power consumption and size of DSP systems.

A further object of the present invention is to provide for shared and reconfigurable memory in order to reduce I/O processor requirements for digital signal processing in processor and co-processor architectures. A further object is to extend a shared, reconfigurable memory architecture to multiple memory blocks and execution units.

Another object of the invention is utilization of novel "bit configuration tables" in connection with shared and reconfigurable memory to use memory more efficiently, and to improve performance by making new data continually available so that the execution unit is never idle. A further object of the invention is to provide improvements in memory addressing methods, architectures and circuits for continuous DSP execution together with simultaneous, continuous Direct Memory Access (DMA) operations.

A still further object of the invention is to provide improvements in execution units for DSP operations, for example execution unit architectures that feature deep-pipeline structures and local registers, and that support parallel operations. Modified execution units can be used to improve efficiency of operation in conjunction with reconfigurable memory.

Yet another object of the present invention is a "virtual two port" memory structure based on a conventional, single-port memory cell. Yet another object is to provide for implementation in both Static Random Access Memory (SRAM) and Dynamic Random Access Memory (DRAM) configurations of the virtual two-port memory.

Summary of the Invention

In view of the foregoing background, the present invention is directed to improved hardware architectures for digital signal processing, and more specifically,

is directed to "memory-centric" methods and apparatus for improved performance in digital signal processing or "DSP" systems. As mentioned above, improvements in DSP performance have been achieved by providing special arithmetic units or "execution units" that are optimized to carry out the arithmetic operations that are commonly required in DSP -- mainly multiplication and addition -- at very high speed. One example of such an execution unit is the "DAU" (data execution unit) provided in the WE DSP32C chip from AT&T. The AT&T execution unit, and others like it, provide relatively fast, floating point arithmetic operations to support computation-intensive applications such as speech, graphics and image processing.

While many improvements have been made in floating-point execution units, pipelined architectures, decreased cycle times, etc., known DSP systems generally work with standard memory systems. For example, DRAM integrated circuits are used for reading input data and, on the output side, for storing output data. DSP data is moved into and out of the DRAM memory systems using known techniques such as multiple-ported memory, DMA hardware, buffers, and the like. While such systems benefit from improvements in memory speed and density, data transfer remains a relative bottleneck. I have reconsidered these known techniques and discovered that significant gains in performance and flexibility can be achieved by focusing on the memory, in addition to the execution unit, and by providing improvements in methods and circuits for moving data efficiently among data sources (such as a host processor bus or I/O channel), memory subsystems, and execution units. Since the focus is on the memory, I coined the term "memory-centric" computing.

One aspect of the invention is a memory subsystem that is partitioned into two or more blocks of memory space. One block of the memory communicates with an I/O or DMA channel to load data, while the other block of memory simultaneously communicates with one or more execution units that carry out arithmetic operations on data in the second block. Results are written back to the second block of memory. Upon conclusion of that process, the memory blocks are effectively "swapped" so that the second block, now holding processed (output) data, communicates with the I/O channel to output that data, while the execution unit communicates with the first

block, which by then has been filled with new input data. Methods and apparatus are shown for implementing this memory swapping technique in real time so that the execution unit is never idle.

Another aspect of the invention provides for interfacing two or more address generators to the same block of memory, so that memory block swapping can be accomplished without the use of larger multi-ported memory cells.

The present invention is useful in a wide variety of signal processing applications including programmable MPEG encode and decode, graphics, speech processing, image processing, array processors, etc. In telecommunications, the invention can be used, for example, for switching applications in which multiple I/O channels are operated simultaneously.

A further aspect of the invention provides for partitioning the memory space into two or more memory "segments" -- with the ability to selectively assign one or more such segments to form a required block of memory. Thus, for example, one block of memory can be configured to include say, four segments, and be associated with an execution unit, while another block of memory is configured to include only one segment and may be assigned to an I/O or DMA channel. This flexibility is useful in matching the memory block size to the requirements of an associated execution unit for a particular operation, say a recursive type of digital filter such as an Infinite Impulse Response (IIR) filter. Memory segments can be of arbitrary size as will be shown in more detail later.

Importantly, the memory is readily "reconfigurable" so that it can adapt to the particular calculations required. Several implementations are disclosed herein. In one embodiment, the memory reconfiguration is controlled by configuration control signals. The configuration control signals may be generated based upon "configuration bits" which can be downloaded from a core processor, instruction decoder, or durable memory, for reconfiguring the memory responsive to the task at hand. In another arrangement, the configuration bits are stored in extended bit positions in the regular memory, so that pointers or traps can be used in software to reconfigure the hardware. A further aspect of the invention provides for generating

configuration control signals in an improved address generator or in a novel Memory-centric DSP Controller ("MDSPC"). The new address generation techniques include both reconfiguring and addressing the memory to support a particular computation in the execution unit.

Another feature of the invention is that memory blocks can be reconfigured both in depth, i.e. number or words or rows, as well as in width (word size). This flexibility simplifies, and speeds, memory I/O for various applications, and provides great flexibility in a single DSP system, which can be implemented as a separate "co-processor" or "on-board" with a general purpose or other core processor. For example, the memory word size can be easily configured to match that of the I/O channel currently in use. The invention can be implemented in both von Neumann as well as Harvard architectures.

A further aspect of the invention, again directed to improvements in data flow, provides ways to interface multiple blocks of memory, in combination with one or more execution units. In some applications, parallel execution units can be used to advantage. Another aspect of the invention is a system that is readily configurable to take advantage of the available execution resources. The configuration bits described above also can include controls for directing data to and from multiple execution units as and when appropriate.

The invention further anticipates an execution unit that can be reconfigured in several ways, including selectable depth (number of pipeline stages) and width (i.e. multiple word sizes concurrently). Preferably the pipelined execution unit(s) includes internal register files with feedback. The execution unit configuration and operation also can be controlled by execution unit configuration control signals. The execution unit configuration control signals can be determined by "configuration bits" stored in the memory, or stored in a separate "configuration table". The configuration table can be downloaded by the host core processor, and/or updated under software control. Preferably, the configuration control signals are generated by the MDSPC controller mentioned above executing microcode. This combination of reconfigurable memory, together with reconfigurable execution units, and the associated techniques

for efficiently moving data between them, provides an architecture that is highly flexible. Microcoded software can be used to take advantage of this architecture so as to achieve new levels of performance in DSP systems. Because the circuits described herein require only one-port or two-port memory cells, they allow higher density and the associated advantages of lowered power dissipation, reduced capacitance, etc. in the preferred integrated circuit embodiments, whether implemented as a stand-alone coprocessor, or together with a standard processor core, or by way of modification of an existing processor core design. An important feature of the architectures described herein is that they provide a tightly coupled relationship between memory and execution units. This feature provides the advantages of reducing internal interconnect requirements, thereby lowering power consumption. In addition, the invention provides for doing useful work on virtually all clock cycles. This feature minimizes power consumption as well.

The foregoing and other objects, features and advantages of the invention will become more readily apparent from the following detailed description of a preferred embodiment of the invention which proceeds with reference to the accompanying drawings.

Brief Description of the Drawings

FIG. 1 is a system level block diagram of an architecture for digital signal processing (DSP) using shared memory according to the present invention.

FIG. 2 illustrates circuitry for selectively coupling two or moare address generators to a single block of memory.

FIG. 3 is a block diagram illustrating portions of the memory circuitry and address generators of Fig. 1 in a fixed-partition memory configuration.

FIG. 4 shows more detail of address and bit line connections in a two-port memory system of the type described.

FIGS. 5A-5C illustrate selected address and control signals in a Processor Implementation of a DSP system, *i.e.* a complete DSP system integrated on a single chip.

FIG. 6A illustrates an alternative embodiment in which a separate DSP program counter is provided for accessing the memory.

FIG. 6B illustrates an alternative embodiment in which an MDSPC accesses the memory.

FIGS. 7A-B are block diagrams that illustrate embodiments of the invention in a Harvard architecture.

FIG. 8 is a conceptual diagram that illustrates a shared, reconfigurable memory architecture according to the present invention.

FIG. 9 illustrates connection of address lines to a shared, reconfigurable memory with selectable (granular) partitioning of the reconfigurable portion of the memory.

FIG. 10 illustrates a system that implements a reconfigurable segment of memory under bit selection table control.

FIG. 11A is a block diagram illustrating an example of using single-ported RAM in a DSP computing system according to the present invention.

FIG. 11B is a table illustrating a pipelined timing sequence for addressing and accessing the one-port memory so as to implement a "virtual two-port" memory.

FIG. 12 illustrates a block of memory having at least one reconfigurable segment with selectable write and read data paths.

FIG. 13A is a schematic diagram showing detail of one example of the write selection circuitry of the reconfigurable memory of Fig. 12.

FIG. 13B illustrates transistor pairs arranged for propagating or isolating bit lines as an alternative to transistors 466 in Fig. 13A or as an alternative to the bit select transistors 462, 464 of Fig. 13A.

FIG. 14 is a block diagram illustrating extension of the shared, reconfigurable memory architecture to multiple segments of memory.

FIG. 15 is a simplified block diagram illustrating multiple reconfigurable memory segments with multiple sets of sense amps.

FIGS. 16A-16D are simplified block diagrams illustrating various examples of memory segment configurations to form memory blocks of selectable size.

FIG. 17 is a block diagram of a DSP architecture illustrating a multiple memory block to multiple execution unit interface scheme in which configuration is controlled via specialized address generators.

FIGS. 18A-18C are simplified block diagrams illustrating various configurations of segments of a memory block into association with multiple execution units.

FIG. 19 is a simplified block diagram illustrating a shared, reconfigurable memory system utilizing common sense amps.

FIG. 20 is a simplified block diagram illustrating a shared, reconfigurable memory system utilizing multiple sense amps for each memory segment.

FIG. 21 is a timing diagram illustrating memory swapping cycles.

FIG. 22A is a block diagram illustrating memory swapping under bit table control.

FIG. 22B is a block diagram illustrating memory swapping under MDSPC control.

FIG. 23 is a simplified block diagram of an MPEG encoder/decoder architecture according to the present invention.

FIG. 24 is a simplified block diagram of a digital signal processing ring topology architecture according to the invention.

Detailed Description of Preferred Embodiment

FIGURE 1

Fig. 1 is a system-level block diagram of an architecture for memory and computing-intensive applications such as digital signal processing. In Fig. 1, a microprocessor interface 40 includes a DMA port 42 for moving data into a memory via path 46 and reading data from the memory via path 44. Alternatively, a single, bi-directional port could be used. The microprocessor interface 40 generically represents an interface to any type of controller or microprocessor. The interface partition indicated by the dashed line 45 in Fig. 1 may be a physical partition, where

the microprocessor is in a separate integrated circuit, or it can merely indicate a functional partition in an implementation in which all of the memory and circuitry represented in the diagram of Fig. 1 is implemented on board a single integrated circuit. Other types of partitioning, use of hybrid circuits, etc., can be used. The microprocessor interface (DMA 42) also includes control signals indicated at 52. The microprocessor or controller can also provide microcode (not shown) for memory control and address generation, as well as control signals for configuration and operation of the functional execution units, as described later.

Because the present invention may be integrated into an existing processor or controller core design, so that both the core processor and the present invention reside in the same integrated circuit, reference will be made herein to the core processor meaning the processor that the present invention has been attached to or integrated with.

In Fig. 1, a two-port memory comprises the first memory block 50, labeled "A" and a second memory block 60, labeled "B." The memory is addressed by a source address generator 70 and a destination address generator 80. A functional execution unit 90 also is coupled to the two-port memory, left and right I/O channels, as illustrated at block B. Preferably, these are not conventional two-port memory I/O ports; rather, they have novel structures described later.

In operation, the interface 44, 46 to the two-port memory block A is a DMA interface that is in communication with the host processor or controller 40. Block A receives data coefficients and optionally other parameters from the controller, and also returns completed data to the controller that results from various DSP, graphics, MPEG encode/decode or other operations carried out in the execution unit 90. This output data can include, for example, FFT results, or convolution data, or graphics rendering data, etc. Thus the single memory can alternately act as both a graphics frame buffer and a graphics computation buffer memory.

Concurrently, the memory block "B" (60) interfaces with the functional execution unit 90. The functional execution unit 90 receives data from the two-port memory block B and executes on it, and then returns results ("writeback") to the

same two-port memory structure. The source address generator 70 supplies source or input data to the functional execution unit while the destination address generator 80 supplies addresses for writing results (or intermediate data) from the execution unit to the memory. Put another way, source address generator 70 provides addressing while the functional execution unit is reading input data from memory block B, and the destination address generator 80 provides addressing to the same memory block B while the functional execution unit 90 is writing results into the memory.

As mentioned above, when the execution unit has completed its work on the data in block B, the memory effectively "swaps" blocks A and B, so that block B is in communication with the DMA channel 42 to read out the results of the execution. Conversely, and simultaneously, the execution unit proceeds to execute on the new input data in block A. This "swapping" of memory blocks includes several aspects, the first of which is switching the memory address generator lines so as to couple them to the appropriate physical block of memory.

In an alternative embodiment, the system can be configured so that the entire memory space (blocks A and B in the illustration) are accessed first by an I/O channel, and then the entire memory swapped to be accessed by the processor or execution unit. In general, any or all of the memory can be reconfigured as described. The memory can be SRAM, DRAM or any other type of random access semiconductor memory or functionally equivalent technology. DRAM refresh is provided by address generators, or may not be required where the speed of execution and updating the memory (access frequency) is sufficient to obviate refresh.

FIGURE 2

Figure 2 illustrates one way of addressing a memory block with two (or more) address generators. Here, one address generator is labeled "DMA" and the other "ADDR GEN" although they are functionally similar. As shown in Fig. 2, one of the address generators 102 has a series of output lines, corresponding to memory word lines. Each output line is coupled to a corresponding buffer (or word line driver or the like), 130 to 140. Each driver has an enable input coupled to a common enable

line 142. The other address generator 104 similarly has a series of output lines coupled to respective drivers 150 to 160. The number of word lines is at least equal to the number of rows of the memory block 200. The second set of drivers also have enable inputs coupled to the common enable control line 142, but note the inverter "bubbles" on drivers 130 to 140, indicating the active-low enables of drivers 150 to 160. Accordingly, when the control line 142 is low, the DMA address generator 102 is coupled to the memory 200 row address inputs. When the control line 142 is high, the ADDR GEN 104 is coupled to the memory 200 row address inputs. In this way, the address inputs are "swapped" under control of a single bit. Alternative circuitry can be used to achieve the equivalent effect. For example, the devices illustrated can be tri-state output devices, or open collector or open drain structures can be used where appropriate. Other alternatives include transmission gates or simple pass transistors for coupling the selected address generator outputs to the memory address lines. The same strategy can be extended to more than two address sources, as will be apparent to those skilled in the art in view of this disclosure.

FIGURE 3

Figure 3 is a block diagram illustrating a physical design of portions of the memory circuitry and address generators of Fig. 1 in a fixed-partition configuration. By "fixed partition" I mean that the size of memory block A and the size of memory block B cannot change dynamically. In Fig. 3, the memory block A (50) and block B (60) correspond to the same memory blocks of Fig. 1. The memory itself preferably is dynamic RAM, although static RAM or other solid state memory technologies could be used as well. In memory block B, just two bits or memory cells 62 AND 64 are shown by way of illustration. In a typical implementation, the memory block is likely to include thousands or even millions of rows, each row (or word) being perhaps 64 or more bits wide. A typical memory block using today's technology is likely to be one or two megabytes. The memory blocks need not be of equal size. Neither memory depth nor word size is critical to the invention.

Two bits are sufficient here to illustrate the concept without unduly complicating the drawing. The source address generator 70 is coupled to both memory blocks A and B. In block B, the top row includes a series of cells including bit cell 62. In fact, the source address generator preferably has output lines coupled to all of the rows of not only block B, but block A as well, although only one row line is illustrated in block A. Note also that corresponding address lines from the AG 70 and the DMA 102 are shown as connected in common, e.g. at line 69. However, in practice, these address lines are selectable as described above with reference to Fig. 2.

A destination address generator 80 similarly is coupled to the row lines of both blocks of memory. Memory cells 62 and 64 are full two-ported cells on the same column in this example. Thus, either source AG 70 or DMA 102 address the left port, while either destination AG 80 or DMA 100 address the right port. A write select multiplexer 106 directs data either from the DMA (42 in Fig. 1) (or another block of memory) or from the execution unit 90, responsive to a control signal 108. The control signal is provided by the controller or microprocessor of Fig, 1, by a configuration bit, or by an MDSPC. The selected write data is provided to column amplifiers 110, 112 which in turn are connected to corresponding memory cell bit lines. 110 and 112 are bit and /bit ("bit bar") drivers. Below cell 64 is a one-bit sense amplifier 116. A bit output from the sense amp 116 is directed, for example, to a latch 72. Both the DMA and the execution unit are coupled to receive data from latch 72, depending on appropriate control, enable and clock signals (not shown here). Or, both the DMA and the execution path may have separate latches, the specifics being a matter of design choice. Only one sense amp is shown for illustration, while in practice there will be at least one sense amp for each column. Use of multiple sense amps is described later.

FIGURE 4

Fig. 4 shows more detail of the connection of cells of the memory to source and destination address lines. This drawing shows how the source address lines

(when asserted) couple the write bit line and its complement, i.e. input lines 110,112 respectively, to the memory cells. The destination address lines couple the cell outputs to the read bit lines 114, 115 and thence to sense amp 116. Although only one column is shown, in practice write and read bit lines are provided for each column across the full width of the memory word. The address lines extend across the full row as is conventional.

FIGURES 21, 22A AND 22B

Timing

Fig. 21 is a conceptual diagram illustrating an example for the timing of operation of the architecture illustrated in Fig. 1. T0A, T1A, etc., are specific instances of two operating time cycles T0 and T1. The cycle length can be predetermined, or can be a parameter downloaded to the address generators. T0 andT1 are not necessarily the same length and are defined as alternating and mutually exclusive, i.e. a first cycle T1 starts at the end of T0, and a second cycle T0 starts at the end of the first period T1, and so on. Both T0 and T1 are generally longer than the basic clock or memory cycle time.

Fig. 22A is a block diagram of a single port architecture which will be used to illustrate an example of functional memory swapping in the present invention during repeating T0 and T1 cyles. Execution address generator 70 addresses memory block A (50) during T0 cycles. This is indicated by the left (T0) portion of AG 70. During T1 cycles, execution address generator 70 addresses memory block B (60), as indicated by the right portion of 70. During T1, AG 70 also receives setup or configuration data in preparation for again addressing Mem Block A during the next T0 cycle. Similarly, during T0, AG 70 also receives configuration data in preparation for again addressing Mem Block B during the next T1 cycle.

DMA address generator 102 addresses memory block B (60) during T0 cycles. This is indicated by the left (T0) portion of DMA AG 102. During T1 cycles, DMA address generator 102 addresses memory block A (50), as indicated by the right portion of 102. During T1, DMA AG 102 also receives setup or configuration data in preparation for again addressing Mem Block B during the next

T0 cycle. Similarly, during T0, DMA 102 also receives configuration data in preparation for again addressing Mem Block A during the next T1 cycle.

During a T0 cycle, the functional execution unit (90 in Fig. 1) is operating continuously on data in memory block A 50 under control of execution address generator 70. Simultaneously, DMA address generator 102 is streaming data into memory block B 60.

At the beginning of a T1 cycle, memory blocks A and B effectively swap such that execution unit 90 will process the data in memory block B 60 under control of execution address generator 70 and data will stream into memory block A 50 under control of DMA address generator 102. Conversely, at the beginning of a T0 cycle, memory blocks A and B again effectively swap such that execution unit 90 will process the data in memory block A 50 under control of execution address generator 70 and data will stream into memory block B 60 under control of DMA address generator 102.

In Fig. 22B, the functions of the execution address generator and DMA address generator are performed by the MDPSC 172 under microcode control.

FIGURES 5A-C

Processor Implementation

The preferred architecture for implementation in a processor application, as distinguished from a coprocessor application, is illustrated in Figs. 5A-C. In Fig. 5A, a two-port memory again comprises a block A (150) and a block B (160). Memory block B is coupled to a DSP execution unit 130. An address generator 170 is coupled to memory block B 160 via address lines 162. In operation, as before, the address generator unit is executing during a first cycle T0 and during time T0 is loading parameters for subsequent execution in cycle T1. The lower memory block A is accessed via core processor data address register 142A or core processor instruction address register 142B. Thus, in this illustration, the data memory and the instructional program memory are located in the same physical memory. A microprocessor system of the Harvard architecture has separate physical memory for

data and instructions. The present invention can be used to advantage in the Harvard architecture environment as well, as described below with reference to Figs. 7A and 7B.

Bit Configuration Tables

Fig. 5A also includes a bit configuration table 140. The bit configuration table can receive and store information from the memory 150 or from the core processor, via bus 180, or from an instruction fetched via the core processor instruction address register 142B. Information is stored in the bit configuration table during cycle T0 for controlling execution during the next subsequent cycle T1. The bit configuration table can be loaded by a series of operations, reading information from the memory block A via bus 180 into the bit configuration tables. This information includes address generation parameters and opcodes. Examples of some of the address parameters are starting address, modulo-address counting, and the length of timing cycles T0 and T1. Examples of op codes for controlling the execution unit are the multiply and accumulate operations necessary for to perform an FFT.

Essentially, the bit configuration table is used to generate configuration control signal 152 which determines the position of virtual boundary 136 and, therefore, the configuration of memory blocks A and B. It also provides the configuration information necessary for operation of the address generator 170 and the DSP execution unit 130 during the T1 execution cycle time. Path 174 illustrates the execution unit/memory interface control signals from the bit configuration table 140 to the DSP execution unit 130. Path 176 illustrates the configuration control signal to the execution unit to reconfigure the execution unit. Path 178 illustrates the op codes sent to executionunit 130 which cause execution unit to perform the operations necessary to process data. Path 188 shows configuration information loaded from the configuration tables into the address generator 170.

The architecture illustrated in Fig. 5A preferably would utilize the extended instructions of a given processor architecture to allow the address register from the

instruction memory to create the information flow into the bit configuration table. In other words, special instructions or extended instructions in the controller or microprocessor architecture can be used to enable this mechanism to operate as described above. Such an implementation would provide tight coupling to the microprocessor architecture.

Memory-centric DSP Controller

Fig. 5B illustrates an embodiment of the present invention wherein the functions of address generator 170 and bit configuration table 140 of Fig. 5A are performed by memory-centric DSP controller (MDSPC) 172. In the embodiment shown in Fig. 5B, the core processor writes microcode for MDSPC 172 along with address parameters into memory block B 150. Then, under core processor control, the microcode and address parameters are downloaded into local memory within MDSPC 172.

A DSP process initiated in MDPSC 172 then generates the appropriate memory configuration control signals 152 and execution unit configuration control signals 176 based upon the downloaded microcode to control the position of virtual boundary 136 and structure execution unit 130 to optimize performance for the process corresponding to the microcode. As the DSP process executes, MDSPC 172 generates addresses for memory block B 160 and controls the execution unit/memory interface to load operands from memory into the execution unit 130 which are then processed by execution unit 130 responsive to op codes 178 sent from MDSPC 172 to execution unit 130. In addition, virtual boundary 136 may be adjusted responsive to microcode during process execution in order to dynamicly optimize the memory and execution unit configurations.

In addition, the MDSPC 172 supplies the timing and control for the interfaces between memory and the execution unit. Further, algorithm coefficients to the execution unit may be supplied directly from the MDSPC. The use of microcode in the MDSPC results in execution of the DSP process that is more efficient than the frequent downloading of bit configuration tables and address parameters associated

with the architecture of Fig. 5A. The microcoded method represented by the MDSPC results in fewer bits to transfer from the core processor to memory for the DSP process and less frequent updates of this information from the core processor. Thus, the core processor bandwidth is conserved along with the amount of bits required to store the control information.

Fig. 5C illustrates an embodiment of the present invention wherein the reconfigurability of memory in the present invention is used to allocate an additional segment of memory, memory block C 190, which permits MDPSC 172 to execute microcode and process address parameters out of memory block C 190 rather than local memory. This saves the time required for the core processor controlled download of microcode and address parameters to local memory in MDSPC 172 that takes place in the embodiment of Fig. 5B. This embodiment requires an additional set of address 192 and data 194 lines to provide the interface between memory block C 190 and MDSPC 172 and address bus control circuitry 144 under control of MDSPC 172 to disable the appropriate address bits from core processor register file 142. This configuration permits simultaneous access of MDSPC 172 to memory block C 190, DSP execution unit 130 to memory block B and the core processor to memory block A.

Similar to the embodiments shown in Figs. 5A and 5B, virtual boundaries 136A and 136B are dynamically reconfigurable to optimize the memory configuration for the DSP process executing in MDSPC 172.

The bit tables and microcode discussed above may alternatively reside in durable store, such as ROM or flash memory. The durable store may be part of memory block A or may reside outside of memory block A wherein the content of durable store is transferred to memory block A or to the address generators or MDSPC during system initialization.

Furthermore, the DSP process may be triggered by either decoding a preselected bit pattern corresponding to a DSP function into an address in memory block A containing the bit tables or microcode required for execution of the DSP function. Yet another approach to triggering the DSP process is to place the bit

tables or microcode for the DSP function at a particular location in memory block A and the DSP process is triggered by the execution of a jump instruction to that particular location. For instance, at system initialization, the microcode to perform a DSP function, such as a Fast Fourier Transform (FFT) or IIR, is loaded beginning at a specific memory location within memory block A. Thereafter, execution of a jump instruction to that specific memory location causes execution to continue at that location thus spawning the DSP process.

FIGURES 6A and 6B

Referring now to Fig. 6A, in an alternative embodiment, a separate program counter 190 is provided for DSP operations. The core controller or processor (not shown) loads information into the program counter 190 for the DSP operation and then that program counter in turn addresses the memory block 150 to start the process for the DSP. Information required by the DSP operations would be stored in memory. Alternatively, any register of the core processor, such as data address register 142A or instruction address register 142B, can be used for addressing memory 150. Bit Configuration Table 140, in addition to generating memory configuration signal 152, produces address enable signal 156 to control address bus control circuitry 144 in order to select the address register which accesses memory block A and also to selectively enable or disable address lines of the registers to match the memory configuration (i.e. depending on the position of virtual boundary 136, address bits are enabled if the bit is needed to access all of memory block A and disabled if block A is smaller than the memory space accessed with the address bit).

Thus, Fig. 6A shows the DSP program counter 190 being loaded by the processor with an address to move into memory block A. In that case, the other address sources in register file 142 are disabled, at least with respect to addressing memory 150. In short, three different alternative mechanisms are illustrated for accessing the memory 150 in order to fetch the bit configurations and other parameters 140. The selection of which addressing mechanism is most advantageous

may depend upon the particular processor architecture with which the present invention is implemented.

Fig. 6B shows an embodiment wherein MDSPC 172 is used to generate addresses for memory block A in place of DSP PC 190. Address enable signal 156 selects between the address lines of MDSPC 172 and those of register file 142 in response to the microcode executed by MDSPC 172. As discussed above, if the microcode for MDSPC 172 resides in memory block A or a portion thereof, MDSPC 172 will be executing out of memory block A and therefore requires access to the content of memory block A.

Memory Arrangement

Referring again to Fig. 5, memory blocks A (150) and B (160) are separated by "virtual boundary" 136. In other words, block A and block B are portions of a single, common memory, in a preferred embodiment. The location of the "virtual boundary" is defined by the configuration control signal generated responsive to the bit configuration table parameters. In this regard, the memory is reconfigurable under software control. Although this memory has a variable boundary, the memory preferably is part of the processor memory, it is not contemplated as a separate memory distinct from the processor architecture. In other words, in the processor application illustrated by Figs. 5 and 6, the memory as shown and described is essentially reconfigurable directly into the microprocessor itself. In such a preferred embodiment, the memory block B, 160, duly configured, executes into the DSP execution unit as shown in Fig. 5.

In regard to Fig. 5B, virtual boundary 136 is controlled based on the microcode downloaded to MDSPC 172. Similarly, in Fig. 5C, microcode determines the position of both virtual boundary 136A and 136B to create memory block C 190.

FIGURES 7A and 7B

Fig. 7A illustrates an alternative embodiment, corresponding to Fig. 5A, of the present invention in a Harvard-type architecture, comprising a data memory block

A 206 and block B 204, and a separate core processor instruction memory 200. The instruction memory 200 in addressed by a program counter 202. Instructions fetched from the instruction memory 200 pass via path 220 to a DSP instruction decoder 222. The instruction decoder in turn provides addresses for DSP operations, table configurations, etc., to an address register 230. Address register 230 in turn addresses the data memory block A 206. Data from the memory passes via path 240 to load the bit configuration tables etc. 242 which in turn configure the address generator for addressing the data memory block B during the next execution cycle of the DSP execution unit 250. Fig. 6 thus illustrates an alternative approach to accessing the data memory A to fetch bit configuration data. A special instruction is fetched from the instruction memory that includes an opcode field that indicates a DSP operation, or more specifically, a DSP configuration operation, and includes address information for fetching the appropriate configuration for the subroutine.

In the embodiment of Fig. 7B, corresponding to the embodiments in Figs. 5B and 5C, MDPSC 246 replaces AG 244 and Bit Configuration Table 242. Instructions in core processor instruction memory 200 that correspond to functions to be executed by DSP Execution Unit 250 are replaced with a preselected bit pattern which is not recognized as a valid instruction by the core processor. DSP Instruction Decode 222 decodes the preselected bit patterns and generates an address for DSP operations and address parameters stored in data memory A and also generates a DSP control signal which triggers the DSP process in MDSPC 246. DSP Instruction Decode 222 can also be structured to be responsive to output data from data memory A 206 into producing the addresses latched in address register 230.

The DSP Instruction Decode 222 may be reduced or eliminated if the DSP process is initiated by an instruction causing a jump to the bit table or microcode in memory block A pertaining to the execution of the DSP process.

To summarize, the present invention includes an architecture that features shared, reconfigurable memory for efficient operation of one or more processors together with one or more functional execution units such as DSP execution units. Fig. 6A shows an implementation of a sequence of operations, much like a

subroutine, in which a core controller or processor loads address information into a DSP program counter, in order to fetch parameter information from the memory. Fig. 6B shows an implementation wherein the DSP function is executed under the control of an MDSPC under microcode control. In Figs. 5A-C, the invention is illustrated as integrated with a von Neumann microprocessor architecture. Figs. 7A and. 7B illustrate applications of the present invention in the context of a Harvard-type architecture. The system of Fig. 1 illustrates an alternative stand-alone or coprocessor implementation. Next is a description of how to implement a shared, reconfigurable memory system.

Reconfigurable Memory Architecture

FIGURE 8

Fig. 8 is a conceptual diagram illustrating a reconfigurable memory architecture for DSP according to another aspect of the present invention. In Fig. 8, a memory or a block of memory includes rows from 0 through Z. A first portion of the memory 266, addresses 0 to X, is associated, for example, with an execution unit (not shown). A second (hatched) portion of the memory 280 extends from addresses from X+1 to Y. Finally, a third portion of the memory 262, extending from addresses Y+1 to Z, is associated, for example, with a DMA or I/O channel. By the term "associated" here we mean a given memory segment can be accessed directly by the designated DMA or execution unit as further explained herein. The second segment 280 is reconfigurable in that it can be switched so as to form a part of the execution segment 266 or become part of the DMA segment 262 as required.

The large vertical arrows in Fig. 8 indicate that the execution portion and the DMA portion of the memory space can be "swapped" as explained previously. The reconfigurable segment 280 swaps together with whichever segment it is coupled to at the time. In this block of memory, each memory word or row includes data and/or coefficients, as indicated on the right side of the figure.

Additional "configuration control bits" are shown to the left of dashed line 267. This extended portion of the memory can be used for storing a bit configuration

table that provides configuration control bits as described previously with reference to the bit configuration table 140 of Figs. 5A and 6A. These selection bits can include write enable, read enable, and other control information. So, for example, when the execution segment 266 is swapped to provide access by the DMA channel, configuration control bits in 266 can be used to couple the DMA channel to the I/O port of segment 266 for data transfer. In this way, a memory access or software trap can be used to reconfigure the system without delay.

The configuration control bits shown in Fig. 8 are one method of effecting memory reconfiguration that relates to the use of a separate address generator and bit configuration table as shown in Figs. 5A and 7A. This approach effectively drives an address configuration state machine and requires considerable overhead processing to maintain the configuration control bits in a consistent and current state.

When the MDSPC of Figs. 5B, 5C and 7B is used, the configuration control bits are unnecessary because the MDSPC modifies the configuration of memory algorithmically based upon the microcode executed by the MDSPC. Therefore, the MDSPC maintains the configuration of the memory internally rather than as part of the reconfigured memory words themselves.

FIGURE 9

Fig. 9 illustrates connection of address and data lines to a memory of the type described in Fig. 8. Referring to Fig. 9, a DMA or I/O channel address port 102 provides sufficient address lines for accessing both the rows of the DMA block of memory 262, indicated as bus 270, as well as the reconfigurable portion of the memory 280, via additional address lines indicated as bus 272. When the block 280 is configured as a part of the DMA portion of the memory, the DMA memory effectively occupies the memory space indicated by the brace 290 and the address lines 272 are controlled by the DMA channel 102. Fig. 9 also shows an address generator 104 that addresses the execution block of memory 266 via bus 284. Address generator 104 also provides additional address lines for controlling the reconfigurable block 280 via bus 272. Thus, when the entire reconfigurable segment

280 is joined with the execution block 266, the execution block of memory has a total size indicated by brace 294, while the DMA portion is reduced to the size of block 262.

The address lines that control the reconfigurable portion of the memory are switched between the DMA address source 102 and address generator 104 via switching means 296. Illustrative switching means for addressing a single block of memory from multiple address generators was described above, for example with reference to Fig. 2. The particular arrangement depends in part on whether the memory is single-ported (see Fig. 2) or multi-ported (see Figs. 3-4). Finally, Fig. 9 indicates data access ports 110 and 120. The upper data port 110 is associated with the DMA block of memory, which, as described, is of selectable size. Similarly, port 120 accesses the execution portion of the memory. Circuitry for selection of input (write) data sources and output (read) data destinations for a block of memory was described earlier. Alternative structures and implementation of multiple reconfigurable memory segments are described below.

It should be noted that the entire block need not be switched *in toto* to one memory block or the other. Rather, the reconfigurable block preferably is partitionable so that a selected portion (or all) of the block can be switched to join the upper or lower block. The granularity of this selection (indicated by the dashed lines in 280) is a matter of design choice, at a cost of additional hardware, e.g. sense amps, as the granularity increases, as further explained later.

FIGURE 10

Fig. 10 illustrates a system that implements a reconfigurable segment of memory 280 under bit selection table control. In Fig. 10, a reconfigurable memory segment 280 receives a source address from either the AG or DMA source address generator 274 and it receives a destination address from either the AG or DMA destination address generator 281. Write control logic 270, for example a word wide multiplexer, selects write input data from either the DMA channel or the execution unit according to a control signal 272. The source address generator 274 includes bit

table control circuitry 276. The configuration control circuitry 276, either driven by a bit table or under microcode control, generates the write select signal 272. The configuration control circuitry also determines which source and destination addresses lines are coupled to the memory -- either "AG" (address generator) when the block 280 is configured as part of the an "AG" memory block for access by the execution unit, or the "DMA" address lines when the block 280 is configured as part of the DMA or I/O channel memory block. Finally, the configuration control logic provides enable and/or clock controls to the execution unit 282 and to the DMA channel 284 for controlling which destination receives read data from the memory output data output port 290.

FIGURE 11

Fig. 11 is a partial block/partial schematic diagram illustrating the use of a single ported RAM in a DSP computing system according to the present invention. In Fig. 11, a single-ported RAM 300 includes a column of memory cells 302, 304, etc. Only a few cells of the array are shown for clarity. A source address generator 310 and destination address generator 312 are arranged for addressing the memory 300. More specifically, the address generators are arranged to assert a selected one address line at a time to a logic high state. The term "address generator" in this context is not limited to a conventional DSP address generator. It could be implemented in various ways, including a microprocessor core, microcontroller, programmable sequencer, etc. Address generation can be provided by a micro-coded machine. Other implementations that provide DSP type of addressing are deemed equivalents. However, known address generators do not provide control and configuration functions such as those illustrated in Fig. 10 -- configuration bits 330. For each row of the memory 300, the corresponding address lines from the source and destination blocks 310, 312, are logically "ORed" together, as illustrated by OR gate 316, with reference to the top row of the memory comprising memory cell 302. Only one row address line is asserted at a given time. For writing to the memory, a multiplexer 320 selects data either from the DMA or from the execution unit,

according to a control signal 322 responsive to the configuration bits in the source address generator 310. The selected data is applied through drivers 326 to the corresponding column of the memory array 300 (only one column, i.e. one pair of bit lines, is shown in the drawing). For each column, the bit lines also are coupled to a sense amplifier 324, which in turn provides output or write data to the execution unit 326 and to the DMA 328 via path 325. The execution unit 326 is enabled by an execution enable control signal responsive to the configuration bits 330 in the destination address block 312. Configuration bits 330 also provide a DMA control enable signal at 332.

The key here is to eliminate the need for a two-ported RAM cell by using a logical OR of the last addresses from the destination and source registers (located in the corresponding destination or source address generators). Source and destination operations are not simultaneous, but operation is still fast. A source write cycle followed by a destination read cycle would take only a total time of two memory cycles.

FIGURE 12

Fig. 12. The techniques and circuits described above for reconfigurable memory can be extended to multiple blocks of memory so as to form a highly flexible architecture for digital signal processing. Fig. 12 illustrates a first segment of memory 400 and a second memory segment 460. In the first segment 400, only a few rows and a few cells are shown for purposes of illustration. One row of the memory begins at cell 402, a second row of the memory begins at cell 404, etc. Only a single bit line pair, 410, is shown for illustration. At the top of the figure, a first write select circuit such as a multiplexer 406 is provided for selecting a source of write input data. For example, one input to the select circuit 406 may be coupled to a DMA channel or memory block M1. A second input to the MUX 406 may be coupled to an execution unit or another memory block M2. In this discussion, we use the designations M1, M2, etc., to refer generically, not only to other blocks of memory, but to execution units or other functional parts of a DSP system in general.

The multiplexer 406 couples a selected input source to the bit lines in the memory segment 400. The select circuit couples all, say 64 or 128 bit lines, for example, into the memory. Preferably, the select circuit provides the same number of bits as the word size.

The bit lines, for example bit line pair 410, extend through the memory array segment to a second write select circuit 420. This circuit selects the input source to the second memory segment 460. If the select circuit 420 selects the bit lines from memory segment 400, the result is that memory segment 400 and the second memory segment 460 are effectively coupled together to form a single block of memory. Alternatively, the second select circuit 420 can select write data via path 422 from an alternative input source. A source select circuit 426, for example a similar multiplexer circuit, can be used to select this input from various other sources, indicated as M2 and M1. When the alternative input source is coupled to the second memory segment 460 via path 422, memory segment 460 is effectively isolated from the first memory segment 400. In this case, the bit lines of memory segment 400 are directed via path 430 to sense amps 440 for reading data out of the memory segment 400. When the bitlines of memory segment 400 are coupled to the second segment 460, sense amps 440 can be sent to a disable or low power standby state, since they need not be used.

FIGURE 13

Fig. 13 shows detail of the input selection logic for interfacing multiple memory segments. In Fig. 13, the first memory segment bit line pair 410 is coupled to the next memory segment 460, or conversely isolated from it, under control of pass devices 466. When devices 466 are turned off, read data from the first memory segment 406 is nonetheless available via lines 430 to the sense amps 440. The input select logic 426 includes a first pair of pass transistors 426 for connecting bit lines from source M1 to bit line drivers 470. A second pair of pass transistors 464 controllably couples an alternative input source M2 bit lines to drivers 470. The pass devices 462, 464, and 466, are all controllable by control bits originating, for

example, in the address generator circuitry described above with reference to Fig. 9. Pass transistors, transmission gates or the like can be considered equivalents for selecting input (write data) sources.

FIGURE 14

Fig. 14 is a high-level block diagram illustrating extension of the architectures of Figs. 12 and 13 to a plurality of memory segments. Details of the selection logic and sense amps is omitted from this drawing for clarity. In general, this drawing illustrates how any available input source can be directed to any segment of the memory under control of the configuration bits.

Fig. 15 is another block diagram illustrating a plurality of configurable memory segments with selectable input sources, as in Fig. 14. In this arrangement, multiple sense amps 482, 484, 486, are coupled to a common data output latch 480. When multiple memory segments are configured together so as to form a single block, fewer than all of the sense amps will be used. For example, if memory segment 0 and memory segment 1 are configured as a single block, sense amp 484 provides read bits from that combined block, and sense amp 482 can be idle.

Figs. 16A through 16D are block diagrams illustrating various configurations of multiple, reconfigurable blocks of memory. As before, the designations M1, M2, M3, etc., refer generically to other blocks of memory, execution units, I/O channels, etc. In Fig. 16A, four segments of memory are coupled together to form a single, large block associated with input source M1. In this case, a single sense amp 500 can be used to read data from this common block of memory (to a destination associated with M1). In Fig. 16B, the first block of memory is associated with resource M1, and its output is provided through sense amp 502. The other three blocks of memory, designated M2, are configured together to form a single block of memory -- three segments long -- associated with resource M2. In this configuration, sense amp 508 provides output from the common block (3xM2), while sense amps 504 and 506 can be idle. Figs. 16C and 16D provide additional examples that are self explanatory in view of the foregoing description. This illustration is not intended to imply that all

memory segments are of equal size. To the contrary, they can have various sizes as explained elsewhere herein.

Fig. 17 is a high-level block diagram illustrating a DSP system according to the present invention in which multiple memory blocks are interfaced to multiple execution units so as to optimize performance of the system by reconfiguring it as necessary to execute a given task. In Fig. 17, a first block of memory M1 provides read data via path 530 to a first execution unit ("EXEC A") and via path 532 to a second execution unit (EXEC B"). Execution unit A outputs results via path 534 which in turn is provided both to a first multiplexer or select circuit MUX-1 and to a second select circuit MUX-2. MUX-1 provides select write data into memory M1.

Similarly, a second segment of memory M2 provides read data via path 542 to execution unit A and via path 540 to execution unit B. Output data or results from execution unit B are provided via path 544 to both MUX-1 and to MUX-2. MUX-2 provides selected write data into the memory block M2. In this way, data can be read from either memory block into either execution unit, and results can be written from either execution unit into either memory block.

A first source address generator S1 provides source addressing to memory block M1. Source address generator S1 also includes a selection table for determining read/write configurations. Thus, S1 provides control bit "Select A" to MUX--1 in order to select execution unit A as the input source for a write operation to memory M1. S1 also provides a "Select A" control bit to MUX-2 in order to select execution unit A as the data source for writing into memory M2.

A destination address generator D1 provides destination addressing to memory block M1. D1 also includes selection tables which provide a "Read 1" control signal to execution A and a second "Read 1" control signal to execution unit B. By asserting a selected one of these control signals, the selection bits in D1 directs a selected one of the execution units to read data from memory M1.

A second source address generator S2 provides source addressing to memory segment M2. Address generator S2 also provides a control bit "select B" to MUX-1

via path 550 and to MUX-2 via path 552. These signals cause the corresponding multiplexer to select execution unit B as the input source for write back data into the corresponding memory block. A second destination address generator D2 provides destination addressing to memory block M2 via path 560. Address generator D2 also provides control bits for configuring this system. D2 provides a read to signal to execution unit A via path 562 and a read to signal to execution unit B via path 564 for selectively causing the corresponding execution unit to read data from memory block M2.

Fig. 18A illustrates at a high level the parallelism of memory and execution units that becomes available utilizing the reconfigurable architecture described herein. In Fig. 18A, a memory block, comprising for example 1,000 rows, may have, say, 256 bits and therefore 256 outputs from respective sense amplifiers, although the word size is not critical. 64 bits may be input to each of four parallel execution units E1 - E4. The memory block thus is configured into four segments, each segment associated with a respective one of the execution units, as illustrated in Fig. 18B. As suggested in the figure, these memory segments need not be of equal size. Fig. 18C shows a further segmentation, and reconfiguration, so that a portion of segment M2 is joined with segment M1 so as to form a block of memory associated with execution unit E1. A portion of memory segment M3, designated "M3/2" is joined together with the remainder of segment M2, designated "M2/2", to form a memory block associated with execution unit E2, and so on.

Note, however, that the choice of one half block increments for the illustration above is arbitrary. Segmentation of the memory may be designed to permit reconfigurability down to the granularity of words or bits if necessary.

FIG. 19.

The use of multiple sense amps for memory segment configuration was described previously with reference to Figs. 15 and 16. Fig. 19 illustrates an alternative embodiment in which the read bit lines from multiple memory segments,

for example read bit lines 604, are directed to a multiplexer circuit 606, or its equivalent, which in turn has an output coupled to shared or common set of sense amps 610. Sense amps 610 in turn provide output to a data output latch 612, I/O bus or the like. The multiplexer or selection circuitry 604 is responsive to control signals (not shown) which select which memory segment output is "tapped" to the sense amps. This architecture reduces the number of sense amps in exchange for the addition of selection circuitry 606.

Fig. 20. is a block diagram illustrating a memory system of multiple configurable memory segments having multiple sense amps for each segment. This alternative can be used to improve speed of "swapping" read data paths and reduce interconnect overhead in some applications.

RAMDSP Design for MPEG-2 Encode/Decode

Introduction

We use "RAMDSP" as a shorthand for embedded DRAM solutions tailored for digital signal processing applications. Full-frame video compression and decompression requires massive amounts of computational power combined with special operations to support efficient use of digital processing engines. Successful single chip designs (without embedded DRAM) include Chromatic Research Mpact2 and C-Cubed DVX in the 3 to 6 million transistors range. The RAMDSP design includes the necessary 8 MB DRAM in the same package and still uses less than 2 million logic transistors.

The RAMDSP solution combines a standard Execution Unit (EU) and three specialized EUs in a multiprocessor arrangement with four separate memories to support MIMD style multiprocessing. The standard EU serves to coordinate activities and is sufficiently powerful to handle the complete decode operation. The specialized EUs are designed to handle the especially compute intensive motion-estimation phase of the MPEG-2 encoder, as well as general signal processing kernels.

MPEG-2 Characteristics

The MPEG-2 *encode* requirements are much more demanding that *decode* requirements. Each frame must be transformed using the Discrete Cosine Transform (DCT) and reference frames (R-frame) must be inverse transformed to give accurate error estimates for encoding intermediate frames (I-frame). The most compute intensive part of the encoding process is to find the "best" motion vector between the R-frame and I-frames, the so-called Motion Estimation (ME) phase. There are a number of algorithms available to find motion vectors, but all require difference kernels, such as sums of absolute differences, or sums of squares of differences.

Specialized MPEG-2 encoder chips often implement highly parallel ME engines, which can perform at rates of 6 to 12 Billion Ops Per Second (Bops). The RAMDSP architecture has been enhanced to directly compute absolute differences in highly parallel SIMD instructions (described below).

RAMDSP / MPEG-2 Configuration

Figure 23 shows an illustrative RAMDSP MPEG-2 configuration. One general RAMDSP block, with 2MB DRAM serves as the supervisor and does general Code/Decode ("Codec") tasks. The other three RAMDSP blocks are specialized for Motion Estimation ("ME Unit"), but can also serve other compute intensive functions. An internal 64-bit bus connects the separate engine/memory units. This bus is also interfaced to the outside world through a custom Bus Interface Unit (BIU). The BIU can include high-speed SRAM buffering and certain specialized functions such as table look-up for bit serial encode/decode.

The general purpose RAMDSP engine has standard 64-bit data paths from the register file through the two ALUs. The specialized ME units have 128-bit data paths. This allows twice the throughput of computational operations, without increasing the code density or complexity. Further specialization includes reduced need for local SRAM and micro-code store due to simplified processing requirements of the ME kernels.

Specialized RAMDSP Features

The RAMDSP design preferably includes several extensions to the Instruction Set Architecture (ISA) that support MPEG-2 algorithms. An absolute difference instruction (PABDIF) allows computing up to 16 absolute differences for 8-bit data in parallel on the standard engine and up to 32 absolute differences on the specialized ME engine, all in a single clock period. These differences can then be accumulated in 16-bit precision in two additional clock periods. Other, more traditional ISAs (e.g., Intel's MMX technology) require three or four instructions to compute the absolute differences and unpack operands for subsequent 16-bit accumulation.

Precision control and rounding, as needed in the DCT algorithm, are supported by a parallel add, round and shift instruction (PARS). This instruction is not usually found in DSP or RISC processor ISAs, but it provides a 3x increase in performance when this operation is required.

Another enhancement to the ISA involves special Adder and Multiplier path instructions that allow interleaving operands to produce a transposition. This primitive is used, for example, in changing from processing rows to columns in the 2D DCT. It affords an increase of 4 to 8 times traditional SIMD designs as illustrated in the following Table:

MPEG-2 Single Chip Solutions

	Chromatic MPACT2	C-CUBE DVX	TRIMEDIA TM-1000	RAMDSP 1+3
Architecture	VLIW ME CoProc	DSP Core RISC Core ME CoProc	VLIW Decode Core	SIMD Vector
Transistors	3.0M	5.5M	5.5M	1.8M
Peak Performance	6 Bops	1 Bop + ME	3.8 Bops	14 Bops
Memory Configuration	4 - 8 MB RAMBUS	8 MB SDRAM	SDRAM	8 MB embedded DRAM

Notes	Toshiba may add 4 MB embedded		MPEG-2 decode only	

RAMDSP Ring Topology

There are numerous ways of arranging and connecting multiple RAMDSP engines, a RISC controller, and one or more Bus Interface Units on a single chip. Many of these methods involve the use a bus to move data between functional units. The disadvantage of using a bus to transfer data is that it requires both time and power. Another method to transfer data between functional units would be to merely switch the memory access configuration such that a different processor has access to the data. Note that this is not really transferring the data at all, but just transferring the right to access it. However, if every processor could access every memory segment, then it does not take very many processors and memory segments before the interconnect would become unwieldy. Thus, it would be of great benefit to have a design that would allow data to be "transferred" by switching memory access and, at the same time, keep the interconnect simple. Hence, the proposed ring topology described below.

The basic idea is to arrange alternating RAMDSP engines (processors) and memory blocks in a circle to form a ring as shown in Figure 24. The example in Figure 24 uses four processors and four memory blocks - this number is arbitrary; it can be scaled. Each memory block consists of, e.g. eight single-ported segments such that multiple segments can be accessed simultaneously in parallel. Through the use of a configuration register, multiplexers, and demultiplexers, several units can access each segment as described above in the description of reconfigurable memory blocks and segments.

Figure 24 shows four bi-directional connections to each memory block: one to each of its two neighboring processors, one to a RISC controller in the center of the ring, and one pointing away from the ring (labeled N, E, S, and W). The four paths pointing away from the ring could either all go to the Bus Interface Unit (BIU)

or to its own off-board interface. The latter would be especially suitable for the case when there are multiple chips on a board arranged in a grid. Also, note that there is control flow between the RISC controller and the BIU for command and status.

Each segment of each memory block need not be able to be switched to all four connections at each block; however, that would be desirable. Each segment could just be able to be attached to two units so long as all six possible pairs are covered. If the processor in the clockwise direction is denoted by CW, the counter clockwise by CCW, the RISC as RISC, and off-board as OFF, then the six combinations are:

1. RISC-OFF
2. RISC-CW
3. RISC-CCW
4. OFF-CW
5. OFF-CCW
6. CW-CCW

The remaining two segments would probably both best be assigned to CW-CCW or else one to OFF-CW and one to OFF-CCW.

Another configuration that should be considered is to rotate the processors 45 degrees counter clockwise on the ring such that a processor and a memory block are integrated (just as in a normal single engine RAMDSP). In this case, one connection to each memory segment is always to its own engine, which will be denoted as LOCAL. Thus, the connections might be:

1. LOCAL-OFF
2. LOCAL-OFF
3. LOCAL-CW
4. LOCAL-CW
5. LOCAL-CCW
6. LOCAL-CCW
7. LOCAL-RISC
8. LOCAL-RISC

Each processor in Figure 24 preferably is a complete RAMDSP engine less the DMEM, which is shown separately. Therefore, each processor has its own program in its own program store. Thus, with on the chip, the architecture is a distributed memory MIMD. This allows tremendous flexibility in how problems can be partitioned. Two simple examples:

1. Consider the case of doing image enhancement on a two-dimensional array of pixel data. The array can be partitioned into strips where each strip is assigned to a given processor in order. At the strip boundaries, each processor needs data from its neighboring strips. With this architecture, the data in the neighboring strips is readily available without having to move it over a bus.

2. Some algorithms partition naturally in into sequential steps. Again consider processing a 2D array of pixel data. In this example, there are a number of steps to the algorithm that can be done in stages. The steps might include, e.g. edge enhancement, feature extraction, registration to another image, compare, etc. A data flow solution works well for problems like this. Stages are assigned to processors in a manner to balance the computational load. For simplicity, assume that there are the same number of stages as there are processors and each stage requires the same amount of computation. In this case, one stage is assigned to each processor and they are assigned clockwise around the ring. As the program runs, each processor reads its input data into its EU from the counter clockwise memory, processes it, and writes out the results to the clockwise memory. Thus, all the processor work in parallel on different stages of the algorithm with the data flowing around the ring without needing any data transfers over a buss.

Evaluation of RAMDSP Technology For Graphics Acceleration

1. Introduction

Prior to 1994, realistic animated 3D graphics were only available on high-end workstations with specialized hardware supporting the required compute- and memory-intensive operations. With the introduction of the Pentium family of processors, and increasingly sophisticated graphics accelerator boards and chips,

animated 3D has arrived in the PC marketplace. The primary driver for this technology on PC's has to date been games; however, as the technology continues to improve, it is expected that more serious applications (real-time simulations, etc.) Will migrate from workstations onto PC's as well.

As of this writing, there are still several awkward bottlenecks in 3D graphics processing on PC platforms. The industry is taking a variety of approaches to reducing these bottlenecks, while still keeping hardware costs low enough to support the mass gaming market. The Stellar RMADSP chip technology, with its parallelism, fast on-chip DRAM, and low power consumption, is potentially the centerpiece processor for a unique and relatively low-cost solution to the bottlenecks in the PC 3D graphics pipeline.

2. The 3D Graphics Pipeline

There are many ways to define (not to mention implement) the 3D graphics pipeline. However, there are two basic conceptual stages: geometry processing and rendering. The operations involved in each of these stages are described in succeeding paragraphs. Note that different authors and implementations present the pipeline differently than it is portrayed here; the intent is simply to survey the important operations with an eye to their implementations on Stellar RAMDSP technology.

2.1 Geometry Processing

Geometry processing is responsible for converting a stored scene into a form ready to be rendered onto the screen; processing takes place on polygons that approximate the surfaces of the scene. This involves transformation of the scene data to reflect the current user viewpoint, removal of hidden surfaces and objects lying outside the field of view, and scaling objects so as to provide realistic perspective in a 3D scene. Geometry processing relies heavily on floating-point operations, and as such is typically performed on the PC itself (due to the superior floating-point performance of Pentium-class processors). The performance of geometry processing is chiefly a function of the number of polygons. Additionally, most operations can be

made SIMD or MIMD parallel, since polygons and their vertices are for most purposes independent of one another.

2.1.1 Mass Storage of 3D Graphics

3D graphics scenes are almost universally stored as polygons in one of several standard file formats; the Drawing Exchange Format (DXF), which has developed for CAD drawings, is a common example. While it is possible in principle to process polygons with any number of edges, virtually all schemes exploit triangles to minimize the mathematical complexity of geometry processing operations. Thus the starting point for geometry processing is generally a list of triangles making up the scene, each triangle being defined by the 3D coordinates of its vertices (in practice, the list of vertices takes the form of a hierarchical tree, to avoid the redundancy of storing and processing shared vertices). Associated with each triangle is its color information; in some cases, precomputed surface normal vectors to the triangles may also be stored in the file format.

2.1.2 Transformation to View Coordinates

Because the stored 3D scene can be viewed from any angle and distance, the first step is to transform it into a coordinate space reflecting the desired view (this is commonly referred to as "camera coordinates"). There are three basic operations used in transformation: translation, scaling, and rotation. Translation simply involves shifting the position of the scene within the coordinate space; it is accomplished by adding the appropriate offset to each coordinate of each vertex. Scaling involves changing the size of the scene within the coordinate space to reflect different viewing distances; it is accomplished by multiplying each coordinate of each vertex by the appropriate scaling factor. Finally, rotation of the scene allows any arbitrary viewpoint to be realized; this requires computation or table-lookup of the sine and cosine of the three angles that the desired viewing direction makes with the stored coordinate axes, in addition to both addition and multiplication. Since vertices are independent of each other from a transformation standpoint, each operation can be made SIMD parallel on different vertices insofar as hardware allows; similarly, the three operations can be made MIMD parallel (that is to say, pipelined).

2.1.3 Illumination

In general, the illumination of a scene is not fixed. Hence the effects of lighting need to be reflected in the color associated with each triangle prior to rendering. Ambient lighting is generally handled by simply scaling the intensity of the color relative to the ambient light level. Point light sources require that the color intensity be scaled according to the cosine of the angle of incidence of light rays with the face of each triangle. Directional lighting, which produces specular effects (e.g., highlighting) is generally ignored in animated applications on PC's as of this writing, since its effects are dependent on both the illumination direction and the viewing direction, requiring considerably more computation. Since triangles are independent of each other from an illumination standpoint, the computation of color intensity for each polygon can be made SIMD parallel insofar as hardware allows.

2.1.4 Perspective Projection

Once the scene has been transformed into camera coordinates, perspective correction must be applied to the constituent polygons to achieve the appearance of 3D depth. This involves multiplying each vertex's x and y coordinate by the viewing distance, and subsequently dividing by its z coordinate. The exponential falloff of illumination with distance is generally ignored, since the effects on the scene are negligible. As with previous operations, perspective projection can be made SIMD parallel insofar as hardware allows.

2.1.5 Object Space Clipping

Object space clipping is the process of discarding those triangles that fall entirely outside the viewing window of the scene as it has been transformed. While it is not strictly necessary to perform this during geometry processing (since during final rendering and rasterization pixels lying outside the field of view will in any case not be projected on the screen; this is known as image-space clipping), it is generally done to reduce the number of triangles that the rendering stage will have to process. Object space clipping involves several multiplications, divides, and comparisons for each vertex; triangles are discarded if they fall outside a six-sided viewing frustum. Triangles that are partially within the viewing frustum may be discarded by re-

tessellation (re-triangularization) at this stage of processing, or they may be retained and clipped during rendering for efficiency's sake. Object-space clipping can be made SIMD parallel on a triangle-by-triangle basis (here vertices belonging to a triangle are not independent of each other).

2.1.6 Backface Culling

Backface culling is the process of discarding those triangles that will be invisible from the current viewpoint, since they lie on the far side of objects in the scene. As with object space clipping, backface culling is performed during geometry processing to reduce the number of triangles that the rendering stage will have to process. Backface culling involves computing the dot product of the surface normal of a triangle face and the view direction vector; if the resulting angle is less than zero (hence obtuse), the triangle face is invisible from the current viewpoint. Backface culling can be made SIMD parallel on a triangle-by-triangle basis.

2.2 Rendering

Once the desired geometry processing steps have been applied to the scene, the resulting triangle list is passed to the rendering stage. Rendering is the process of converting the polygon representation of the scene to a pixel image, ready to be displayed on the screen. Rendering chiefly involves integer operations on pixels, in contrast with the floating-point operations performed during geometry processing, and is in general far more memory-intensive than geometry processing. The bulk of rendering takes place on the graphics accelerator board in most modern PC systems. In addition to the conversion from triangles to pixels, several other operations are most conveniently performed at rendering time; these are described below. As in the geometry processing phase, there are numerous opportunities to exploit both SIMD and MIMD parallelism during rendering.

2.2.1 Shading

Associated with each triangle is a single color, selected based on the color of the stored triangle in conjunction with any lighting effects applied during geometry processing. Simply rasterizing a scene based on such triangles results in a faceted

scene, in which the triangle edges are readily apparent. Shading, in this context, is the process of generating non-uniform color across triangles so as to produce a smoother, more realistic appearance. The most commonly used algorithm for animated graphics is Gouraud shading; another algorithm, Phong shading, yields better results but is computationally too costly to be used in animation. Gouraud shading involves first computing the surface normals at all vertices, by averaging the surface normals of the triangles meeting at any given vertex. Then the dot product of each vertex's normal and the light source is computed, resulting in an intensity value for each vertex. Finally, the intensities of the pixels within triangles are computed by interpolating between the values at the vertices. Shading lends itself to both SIMD and MIMD parallelism, since the vertex normals can be computed in parallel and then passed down the pipeline to the interpolation phase, given appropriate hardware support.

2.2.2 Z-Buffering

Z-buffering is a technique for foolproof hidden surface removal. The hidden surfaces that Z-buffering addresses are those that face the viewing direction, but are obscured by other surfaces closer to the viewer. Z-buffering is generally performed at the time that triangles are converted to pixels. A Z-buffer is conceptually a 2D buffer with the same number of elements as the screen display; the element size is typically 16 or 32 bits. Each x-y element of the buffer is initialized to the maximum z value (depth) of the scene. Then as each triangle is processed, the z values of the points within the triangle are compared to the contents of the Z-buffer at the appropriate x-y location. If the triangle point's z value is less than that contained in the Z-buffer (i.e., closer to the viewer), that element of the Z-buffer is updated with the z value of the triangle point. At the same time, the pixel corresponding to that x-y location is written into the frame buffer. Hence, when all triangles have been rasterized, the frame buffer contains only pixels corresponding to triangle points visible to the viewer; at this point the Z-buffer can be discarded. Given that the surface normal for the triangle has been pre-computed, processing each point requires two multiplies, two additions, a divide, and a compare. Z-buffering is also memory

intensive; for a 1280 X 1024 X 32-bit buffer, over 4 megabytes of RAM are required in addition to the frame buffer and other ancillary data structures. Z-buffering lends itself to SIMD processing in particular, since the end result is the same regardless of the order in which triangles are processed.

2.2.3 Texture Mapping

As a scene becomes more and more detailed, more and more triangles are required to represent it. Since the processing time of the 3D graphics pipeline as a whole is strongly a function of the triangle count, it is desirable to keep the number of triangles to a minimum. Texture mapping is a technique allowing an arbitrary amount of detail to be incorporated into a scene, while still keeping the triangle count reasonable. It also allows certain special effects that would be difficult to achieve with higher triangle density. Texture mapping involves the application of pre-generated patterns or pictures to triangle faces in the scene; a classic example is the application of a brick pattern to a polygon, although there is no requirement that the applied picture be regular like brickwork.

Texture maps are stored as pixel bitmaps; their elements are referred to as texels. In applying a texture map to a given triangle in a scene, one or more techniques are typically used to prevent aliasing artifacts. First, the texture map must be perspective-corrected to conform to the orientation of the triangle surface in 3D space. Second, bilinear filtering may be used to reduce the blockiness that results from adjacent display pixels being determined by a single texel; bilinear filtering simply uses a texel averaging scheme in which the four orthogonally adjacent texels also contribute to the pixel's value. MIP-mapping uses pre-computed texture maps of various resolutions in lieu of bilinear filtering; the texture map applied to a given triangle is then chosen based on the triangle's size. Trilinear filtering is generally employed along with MIP-mapping to smooth out textural discontinuities between adjacent triangles.

Texture mapping is memory-intensive, from both size and bandwidth standpoints. Using the MIP-mapping approach, a number of differently-scaled maps (typically on the order of 6 or 8) must be stored for each triangle in the scene; it is

not unusual for the memory consumed for a complex scene to approach 10 MB. These maps must then be accessed in essentially random order, since there is no a priori order to polygon processing, and the particular map used for a given polygon similarly cannot be known in advance. The memory requirements of texture mapping were in fact one of the key motivators behind Intel's Accelerated Graphics Port (AGP) bus (discussed below).

3. Relevant Technologies

3.1 Intel Accelerated Graphics Port (AGP)

AGP is a new bus design, based on the PCI standard, that is aimed at providing high bandwidth and latency in a dedicated graphics bus, along with a new operating mode aimed specifically at reducing the texture mapping bottleneck. It is just now finding its way onto mainstream motherboards. Ultimately, it will offer up to 266 MHz performance, although current implementations are running at 66 MHz (twice the speed of the PCI bus).

The bus offers two operating modes: DMA and Execution. DMA mode is essentially just a faster, dedicated PCI bus; it is aimed at allowing higher-speed bulk transfers of data back and forth between the host PC RAM and the accelerator board's RAM. Execution mode is specifically designed to accommodate the multiple random accesses to host RAM that are typical of current texture mapping approaches. In Execution mode, the accelerator board maps an area of the host PC's RAM for dedicated usage; thence it can do small transfers to and from the host via direct-addressed transfer. Here, as in DMA mode, the faster speed of the bus is really the advantage offered by AGP; the ability to direct address the host's RAM is chiefly a programming convenience.

3.2 Intel MMX

The MMX extensions to the Pentium instruction set allow an extra level of SIMD parallelism to be applied to typical graphics operations (particularly those of geometry processing). The MMX architecture overloads the Pentium's floating point registers with a second mode, in which 8, 16, and 32-bit parallel arithmetic

operations are possible. This is a genuine performance gain during long sequences of adds and multiples that are characteristic of geometry processing.

4. Benchmarks

There are no industry standard chip-level benchmarks for 3D graphics accelerator chips, because graphics performance is dependent on many things besides raw chip performance: the host PC, the accelerator board logic, the SVGA subsystem, and so forth. There are, however, a number of generally recognized benchmarks at the integrated system level, that primarily reflect the performance of the accelerator board (other things being more or less equal in the overall system configuration). Two key benchmarks are: the Ziff-Davis 3D Winbench, championed by PC Magazine; and Wizmark, pushed by 3Dfx Interactive (the manufacturer of the Voodoo graphics accelerator chip). The former is important because it is vendor-independent, the latter because any new graphics accelerator approach will have to compare itself to the forthcoming Voodoo2-based accelerator boards, widely touted as the state of the art in 1998 (though such boards are not scheduled to ship until March).

That said, all the accelerator chip vendors do issue various numbers intended to reflect their standalone performance. As with all such numbers, these are part quantitative truth and part marketing hype, and one vendor's definition of a number is not necessarily comparable to anothers. In any case, the numbers include billions of operations per second (BOPS), triangles rendered per second, pixels generated per second, and frames rendered per second. In the absence of a defined RAMDSP chip and board architecture, it is difficult to arrive at any meaningful estimates for a RAMDSP-based solution; however, it is believed that RAMDSP could compete favorably in this regard once an architecture was defined.

5. RAMDSP as Graphics Accelerator

A suitably architected RAMDSP chip has the potential to be a uniquely powerful graphics accelerator, for two key reasons: (1) Most or all of the 3D

graphics pipeline could be implemented on-chip, with consequent savings in host PC cycles and memory (both size and bandwidth/latency); and (2) Due to the on-chip DRAM and multi-level parallelism of RAMDSP, several of the bottleneck operations in the 3D pipeline (notably texture mapping) could be handled much more efficiently than in current competing products in the PC marketplace. An additional attractive feature of RAMDSP technology is its low power consumption, making it suitable for powerful 3D graphics on laptops and even PDA's; however, it is unclear at this writing that the demand for animated 3D is strong on such platforms.

Suitably architected, in this context, means the following: The chip must contain multiple EU's and associated DRAM's. The fourfold EU/DRAM ring configuration (with a central RISC executive processor) that has been discussed would likely be adequate to allow the entire pipeline to be chip-resident. It will be obvious to those having skill in the art that many changes may be made to the details of the above-described embodiment of this invention without departing from the underlying principles thereof. The present disclosure will enable those skilled in the art to apply the architectures described here to various designs and implementations. The scope of the present invention should, therefore, be determined only by the following claims.

Claims

1. An embedded DRAM semiconductor integrated circuit architecture for graphics processing comprising:

a plurality of processor engines;

a plurality of memory blocks; and

a RISC processor, all formed on a single integrated circuit in which the memory blocks comprise embedded DRAM;

wherein the processor engines and the memory blocks are alternately arranged so as to form a ring topology; and

the RISC processor has access to each of the memory blocks.

2. An embedded DRAM architecture according to claim 1 wherein the architecture includes at least four of said processor engines and at least four of said memory blocks arranged in the ring topology.

3. An embedded DRAM architecture according to claim 1 wherein each of the processor engines includes a local processor and a program store for storing instructions executable in the corresponding local processor, whereby the described architecture defines a single-chip, distributed MIMD (multiple instruction multiple datapath) system.

4. An embedded DRAM architecture according to claim 1 wherein at least one of the memory blocks includes multiple segments and is reconfigurable so that multiple of such segments are selectively accessible in parallel.

5. An embedded DRAM architecture according to claim 1 wherein said at least one of the memory blocks includes at least eight reconfigurable segments.

6. An embedded DRAM architecture according to claim 1 wherein each of the memory blocks is reconfigurable to provide connection of a selected memory segment to one or more of the other memory segments within the block thereby forming memory sub-blocks of selected size.

7. An embedded DRAM architecture according to claim 5 and further comprising a configuration register for controlling a configuration of at least one of the memory blocks under software control.

8. An embedded DRAM architecture according to claim 5 wherein the configuration register is loadable under control of software executed by the RISC processor.

9. An embedded DRAM architecture according to claim 1 wherein each of the processor engines is associated with a corresponding one of the memory blocks and said corresponding memory block is configurable to provide direct access by the correspondoing processor engine.

10. An embedded DRAM semiconductor integrated circuit architecture for graphics processing comprising:

a plurality of processor engines; and

a plurality of memory blocks, all formed on a single integrated circuit in which the memory blocks comprise embedded DRAM; and

wherein the processor engines and the memory blocks are alternately arranged so as to form a ring topology; and at least one of the reconfigurable memory blocks includes at least two reconfigurable memory segments and the memory block is reconfigurable so as to provide access by a selected one or both of two adjacent processor engines.

11. An embedded DRAM architecture according to claim 9 and further comprising a RISC processor and wherein the said reconfigurable memory block is configurable so as to provide access by the RISC processor, so that the RISC processor has access to the same memory locations as the two adjacent processor engines.

12. An embedded DRAM architecture according to claim 10 wherein at least one of the memory blocks has at least one memory segment having at least four connections that can be established under software control, said connections including a first connection to a first adjacent processor engine, a second connection to a second adjacent processor engine, a third connection to the RISC processor.

13. An embedded DRAM architecture according to claim 11 wherein each segment of the memory block is configurable so as to connect to at least a selected two units among the adjacent processors, the RISC processor and a off-board interface port.

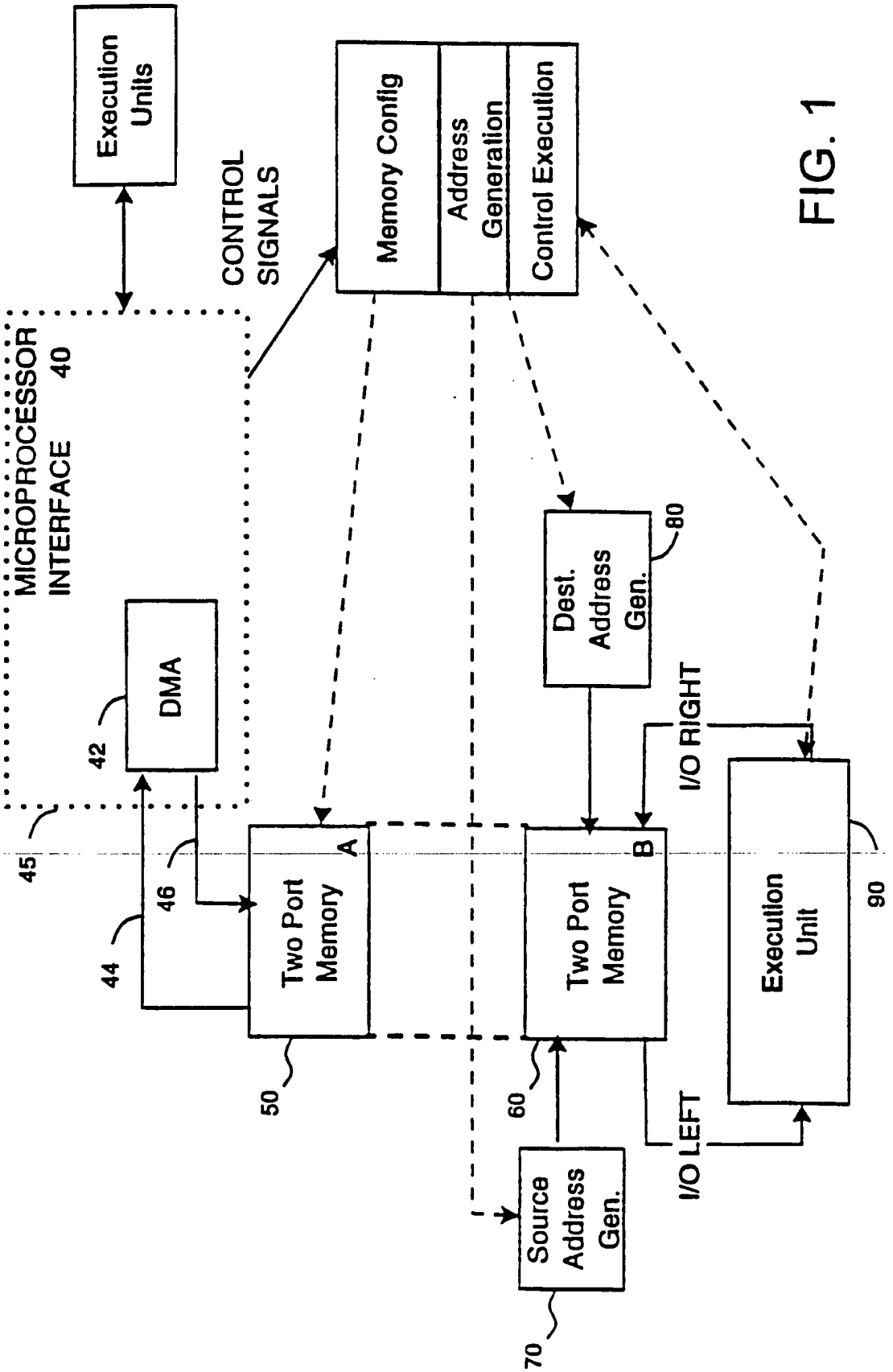

Execution Units
CONTROL SIGNALS
Memory Config
Address Generation
Control Execution
MICROPROCESSOR INTERFACE 40
DMA
42
Dest. Address Gen.
80
I/O RIGHT
45
46
44
Two Port Memory A
Two Port Memory B
Execution Unit
90
50
60
I/O LEFT
Source Address Gen.
70

FIG. 1

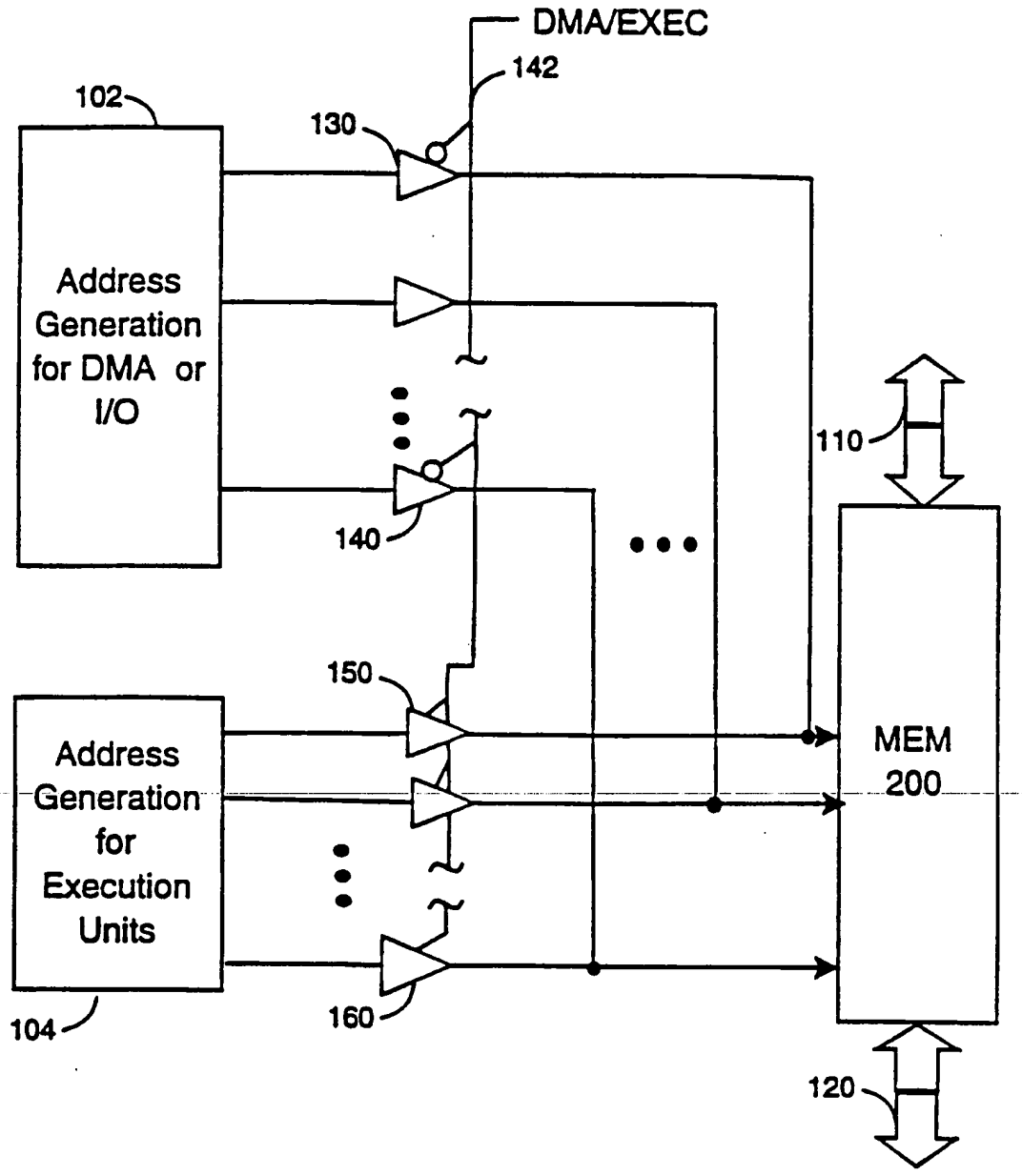

DMA/EXEC
142
102
130
Address Generation for DMA or I/O
110
140
150
Address Generation for Execution Units
MEM 200
104
160
120

FIG. 2

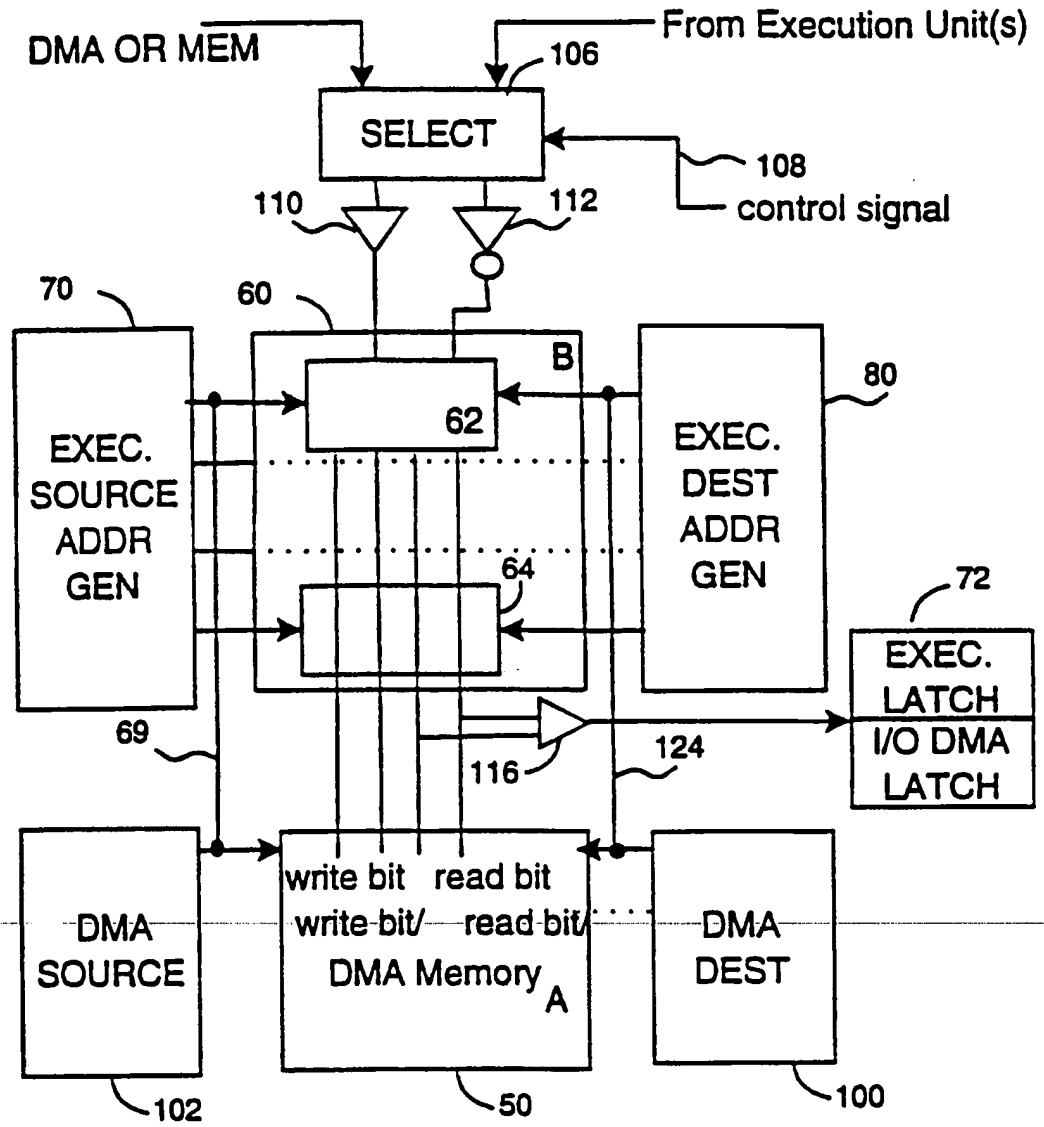

DMA OR MEM
From Execution Unit(s)
106
SELECT
108
control signal
110
112
70
60
B
62
80
EXEC. SOURCE ADDR GEN
EXEC. DEST ADDR GEN
64
72
EXEC. LATCH
I/O DMA LATCH
69
116
124
DMA SOURCE
write bit
read bit
write bit/
read bit/
DMA Memory A
DMA DEST
102
50
100

FIG. 3

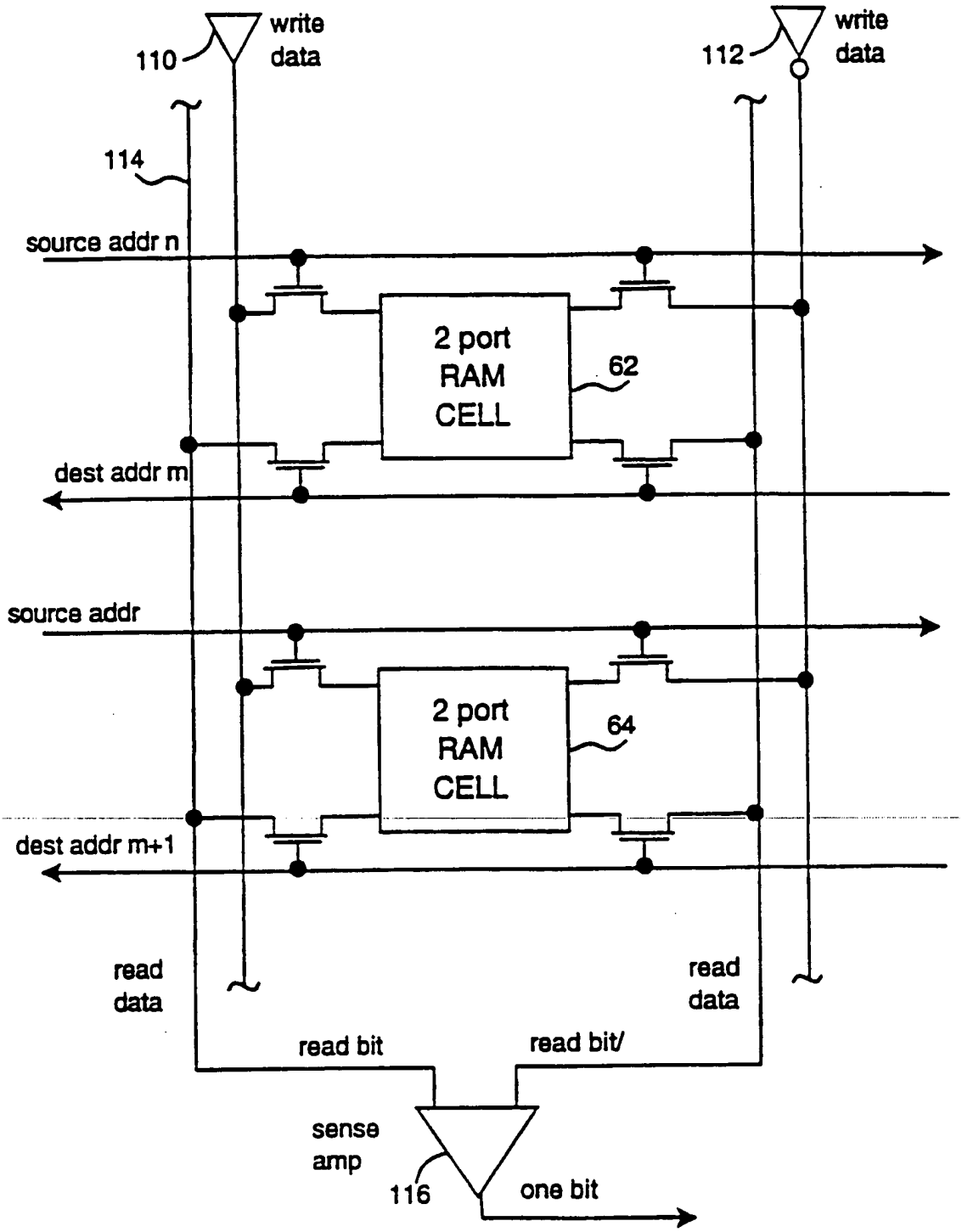

write
data
110
write
data
112
114
source addr n
2 port
RAM
CELL
62
dest addr m
source addr
2 port
RAM
CELL
64
dest addr m+1
read
data
read
data
read bit
read bit/
sense
amp
116
one bit

FIG. 4

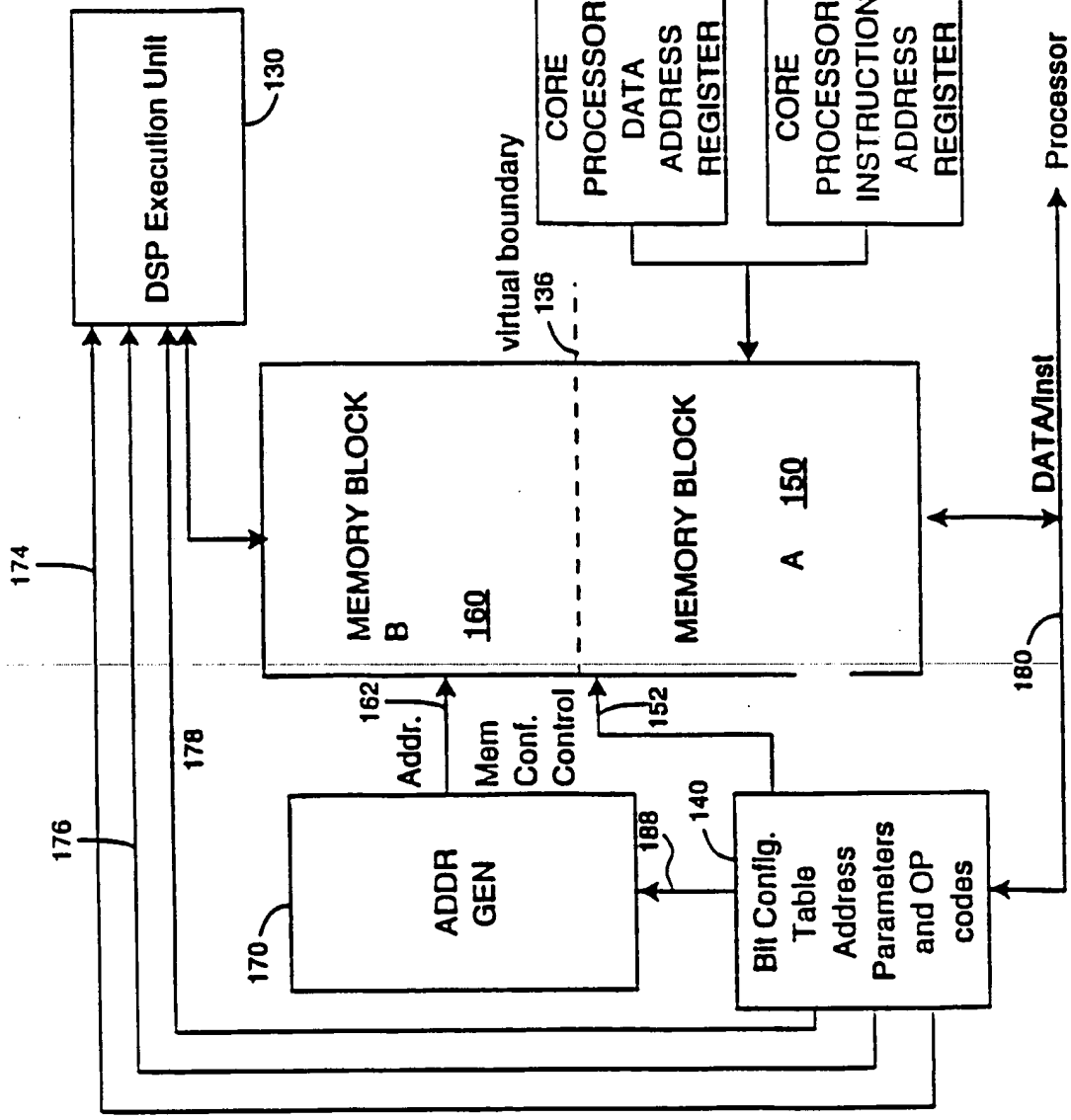

DSP Execution Unit
130
142A
CORE PROCESSOR DATA ADDRESS REGISTER
142B
CORE PROCESSOR INSTRUCTION ADDRESS REGISTER
Processor
virtual boundary
136
MEMORY BLOCK B 160
MEMORY BLOCK A 150
DATA/Inst
174
178
176
162
Addr.
Mem Conf. Control
152
180
ADDR GEN
170
188
140
Bit Config. Table Address Parameters and OP codes

FIG. 5A

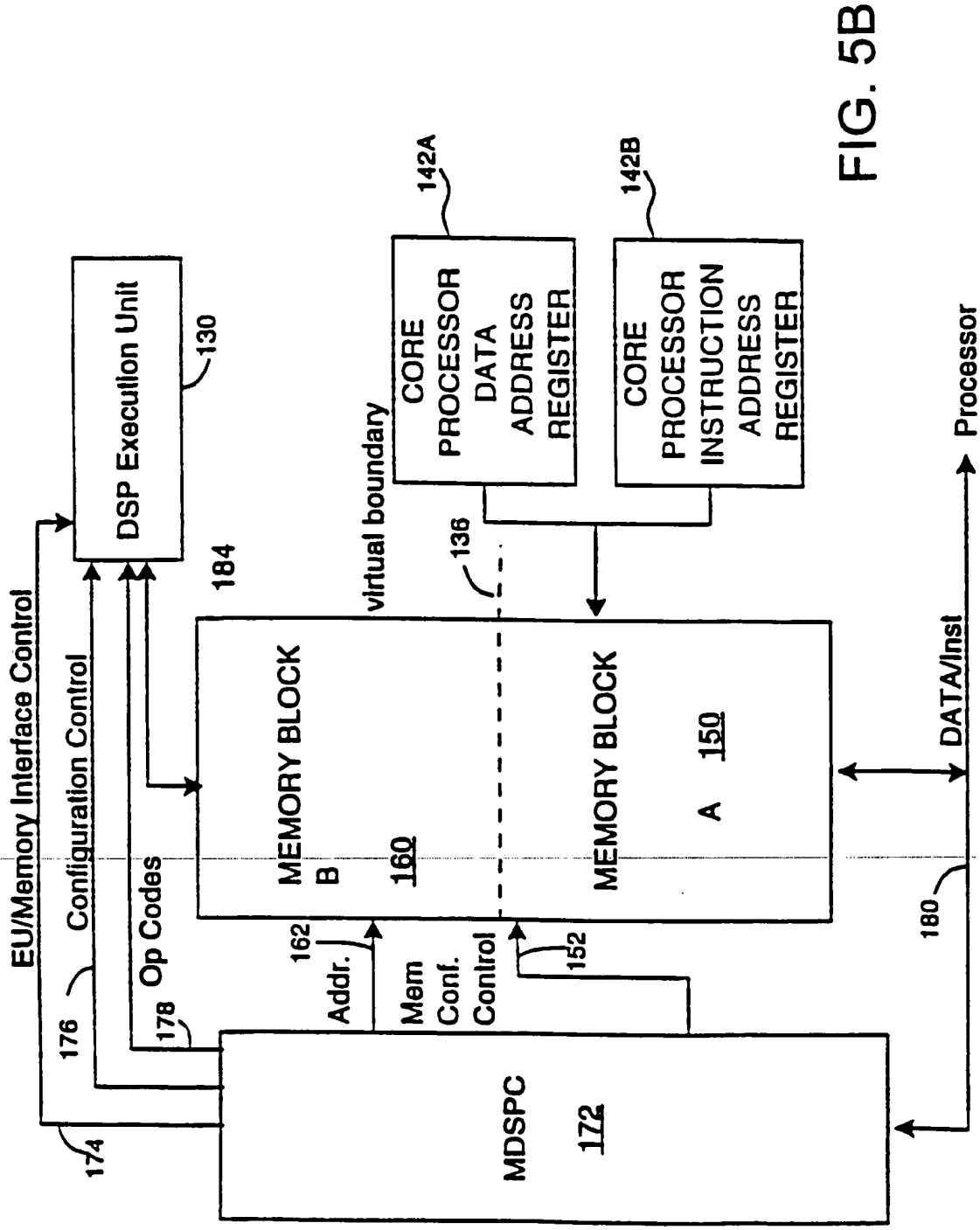

DSP Execution Unit
130
142A
CORE PROCESSOR DATA ADDRESS REGISTER
142B
CORE PROCESSOR INSTRUCTION ADDRESS REGISTER
virtual boundary
136
184
MEMORY BLOCK B 160
MEMORY BLOCK A 150
Processor
DATA/Inst
180
EU/Memory Interface Control
Configuration Control
Op Codes
176
178
174
162
Addr.
Mem Conf. Control
152
MDSPC 172

FIG. 5B

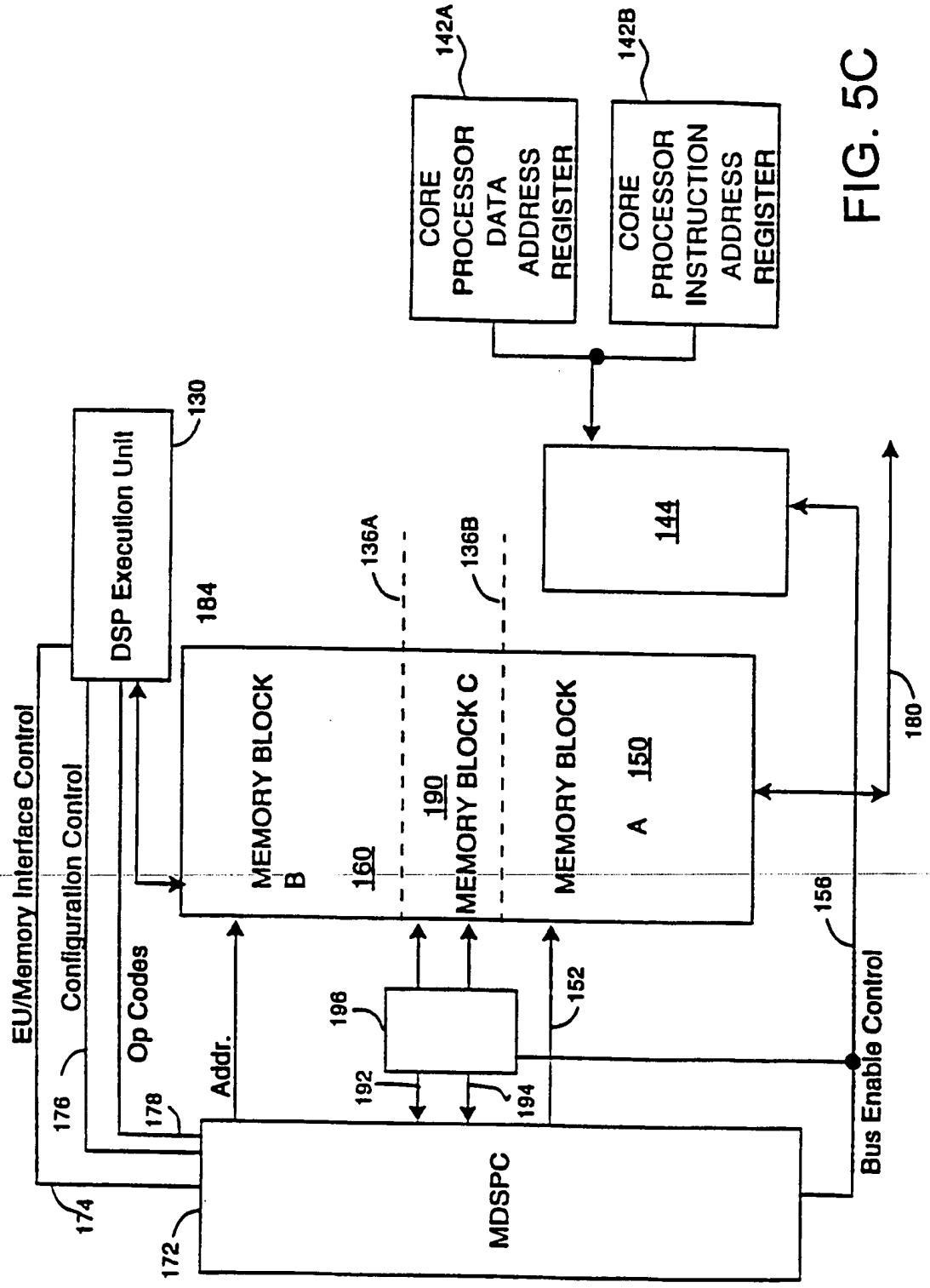

142A
142B
CORE PROCESSOR DATA ADDRESS REGISTER
CORE PROCESSOR INSTRUCTION ADDRESS REGISTER
130
DSP Execution Unit
144
184
136A
136B
MEMORY BLOCK B
160
MEMORY BLOCK C
190
MEMORY BLOCK A
150
180
EU/Memory Interface Control
Configuration Control
Op Codes
Addr.
176
178
174
172
MDSPC
196
192
194
152
156
Bus Enable Control

FIG. 5C

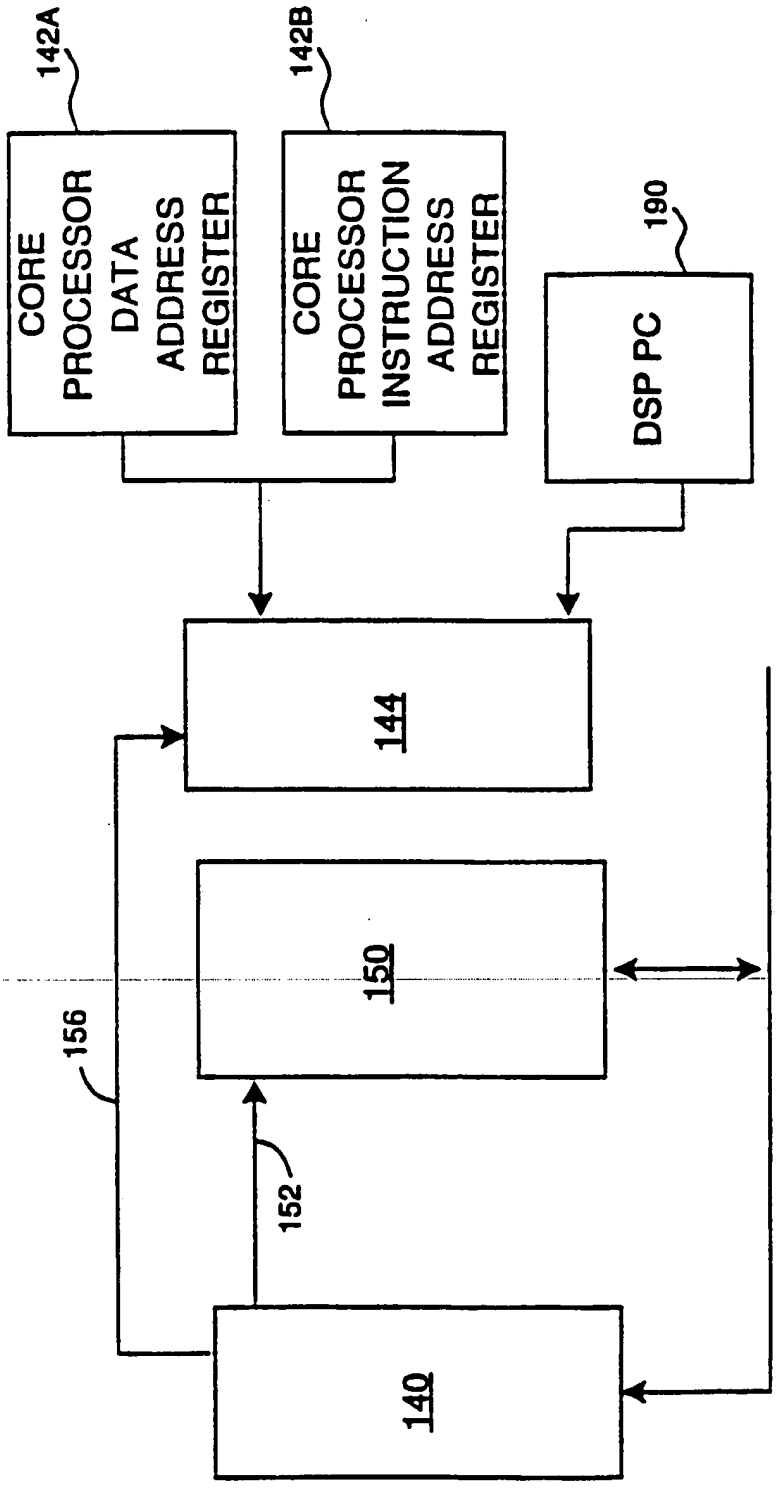

142A
142B
CORE PROCESSOR DATA ADDRESS REGISTER
CORE PROCESSOR INSTRUCTION ADDRESS REGISTER
190
DSP PC
144
150
156
152
140

FIG. 6A

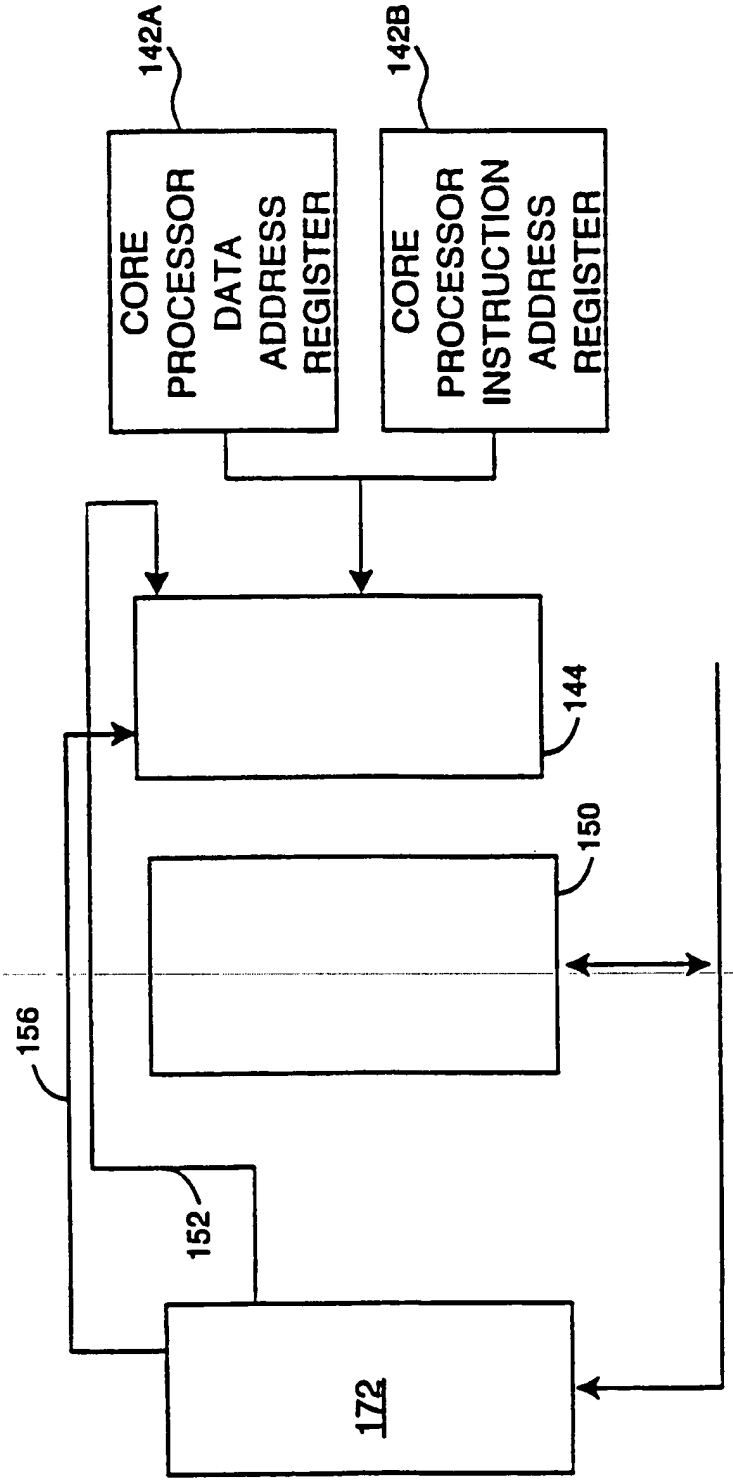

142A
142B
CORE PROCESSOR DATA ADDRESS REGISTER
CORE PROCESSOR INSTRUCTION ADDRESS REGISTER
144
150
156
152
172

FIG. 6B

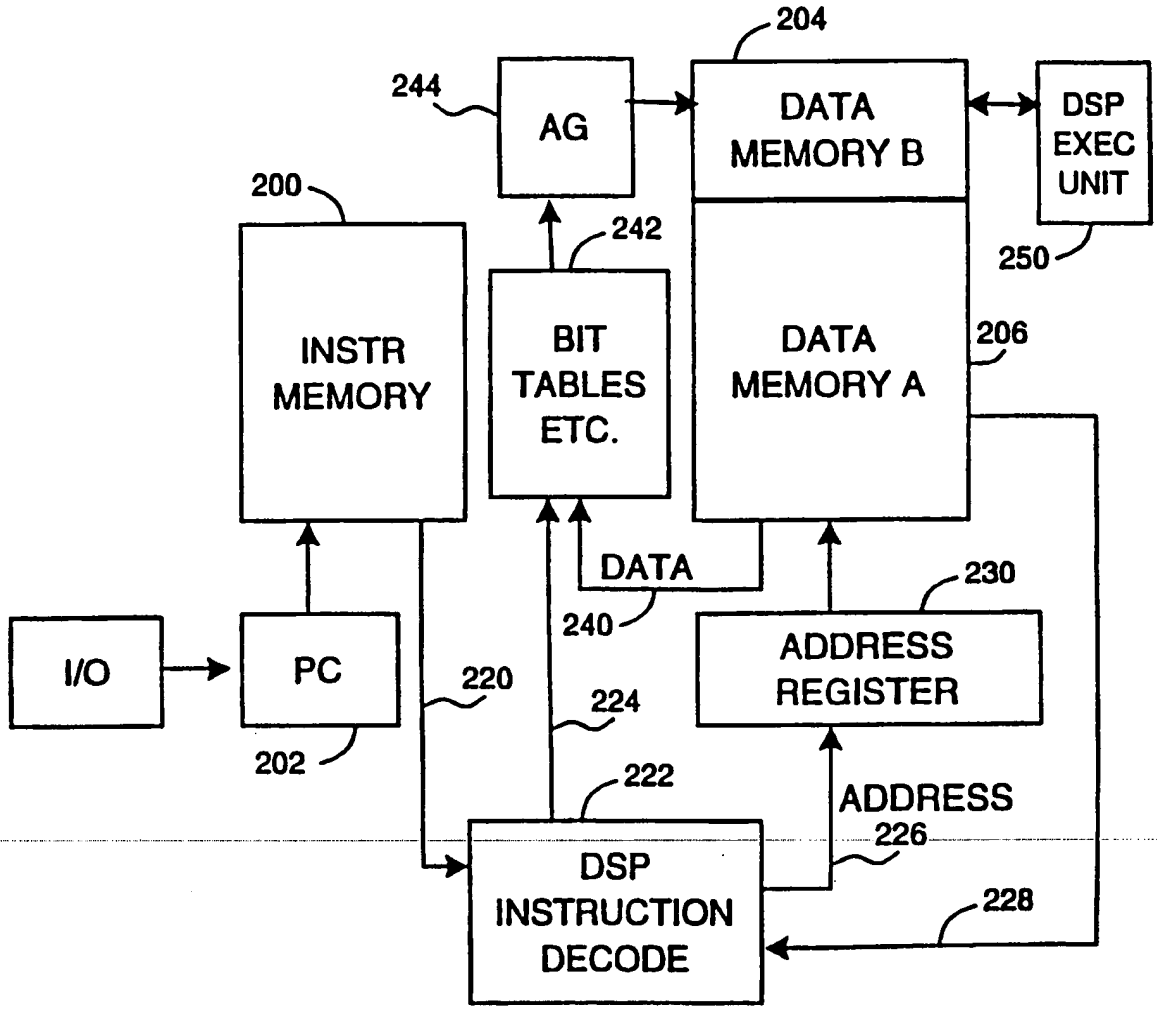

204
244
AG
DATA MEMORY B
DSP EXEC UNIT
250
200
242
INSTR MEMORY
BIT TABLES ETC.
DATA MEMORY A
206
DATA
240
230
I/O
PC
220
224
ADDRESS REGISTER
202
222
ADDRESS
226
DSP INSTRUCTION DECODE
228

FIG. 7A

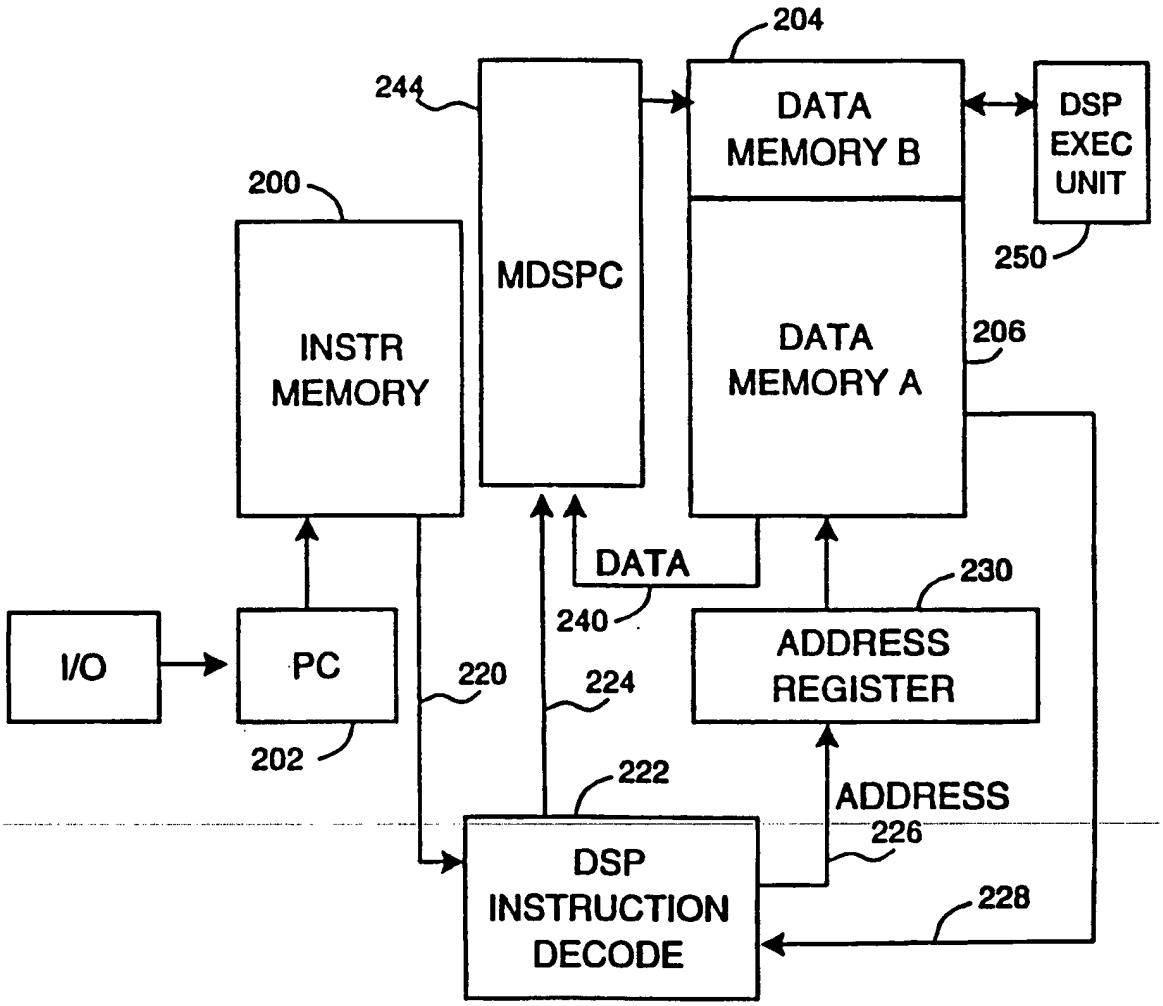

204
244
DATA MEMORY B
DSP EXEC UNIT
250
200
MDSPC
INSTR MEMORY
DATA MEMORY A
206
DATA
240
230
I/O
PC
220
224
ADDRESS REGISTER
202
222
ADDRESS
226
DSP INSTRUCTION DECODE
228

FIG. 7B

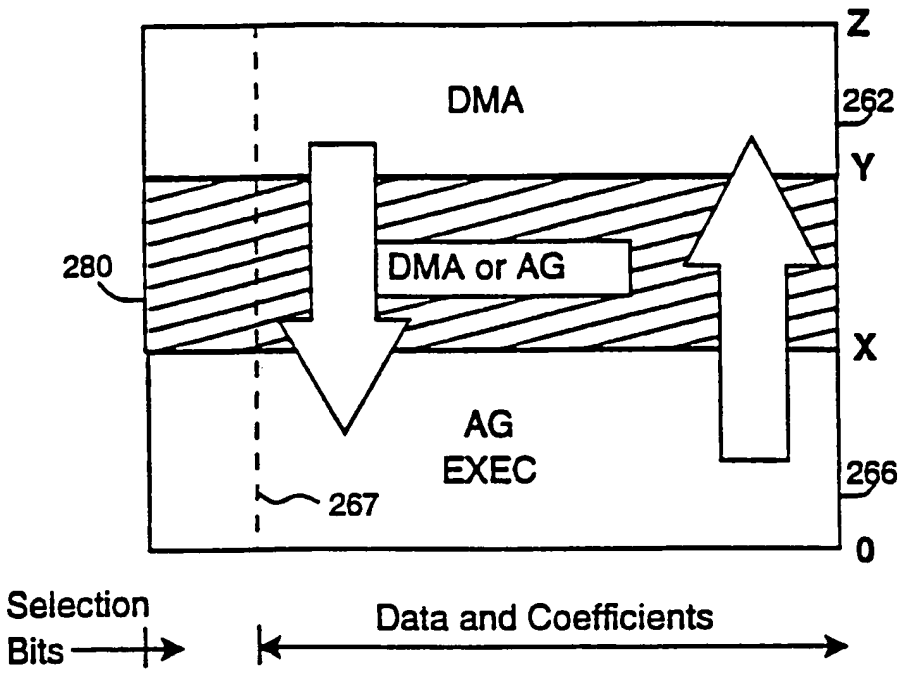

Z
DMA
262
Y
280
DMA or AG
X
AG
EXEC
266
267
0
Selection
Bits
Data and Coefficients

FIG. 8

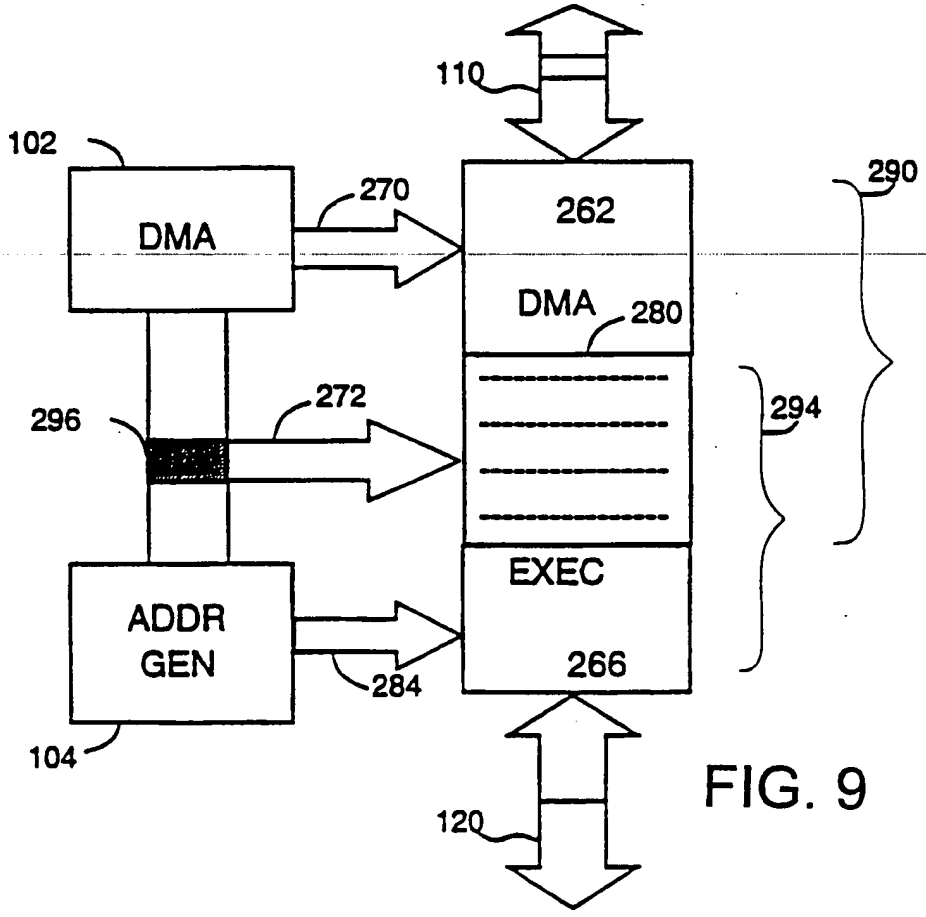

110
102
DMA
270
262
290
DMA
280
296
272
294
EXEC
ADDR
GEN
266
284
104
120

FIG. 9

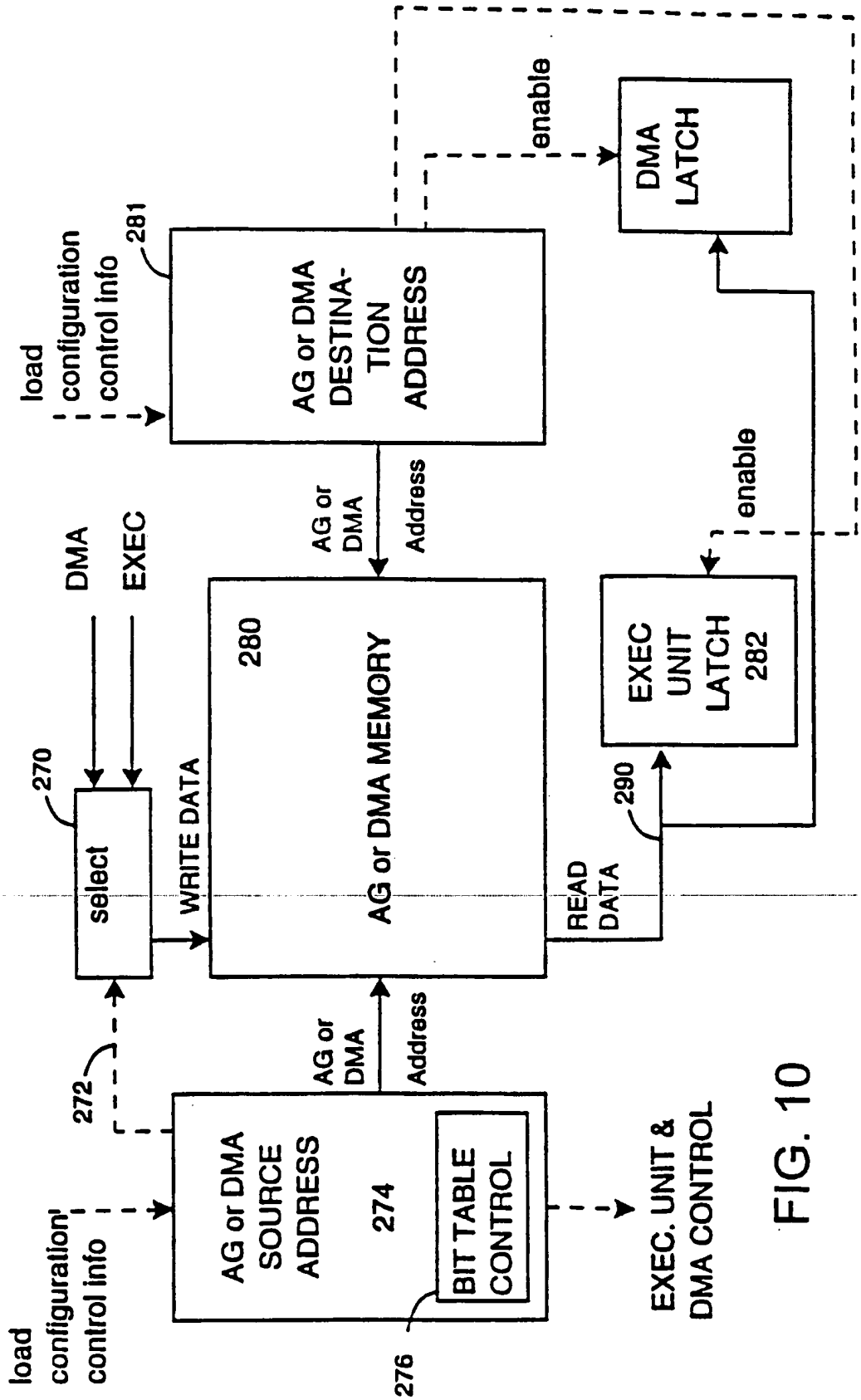

load configuration control info
281
AG or DMA DESTINA-TION ADDRESS
enable
DMA LATCH
AG or DMA Address
DMA
EXEC
280
AG or DMA MEMORY
EXEC UNIT LATCH 282
enable
270
select
WRITE DATA
READ DATA
290
272
AG or DMA Address
load configuration control info
AG or DMA SOURCE ADDRESS 274
BIT TABLE CONTROL
276
EXEC. UNIT & DMA CONTROL

FIG. 10

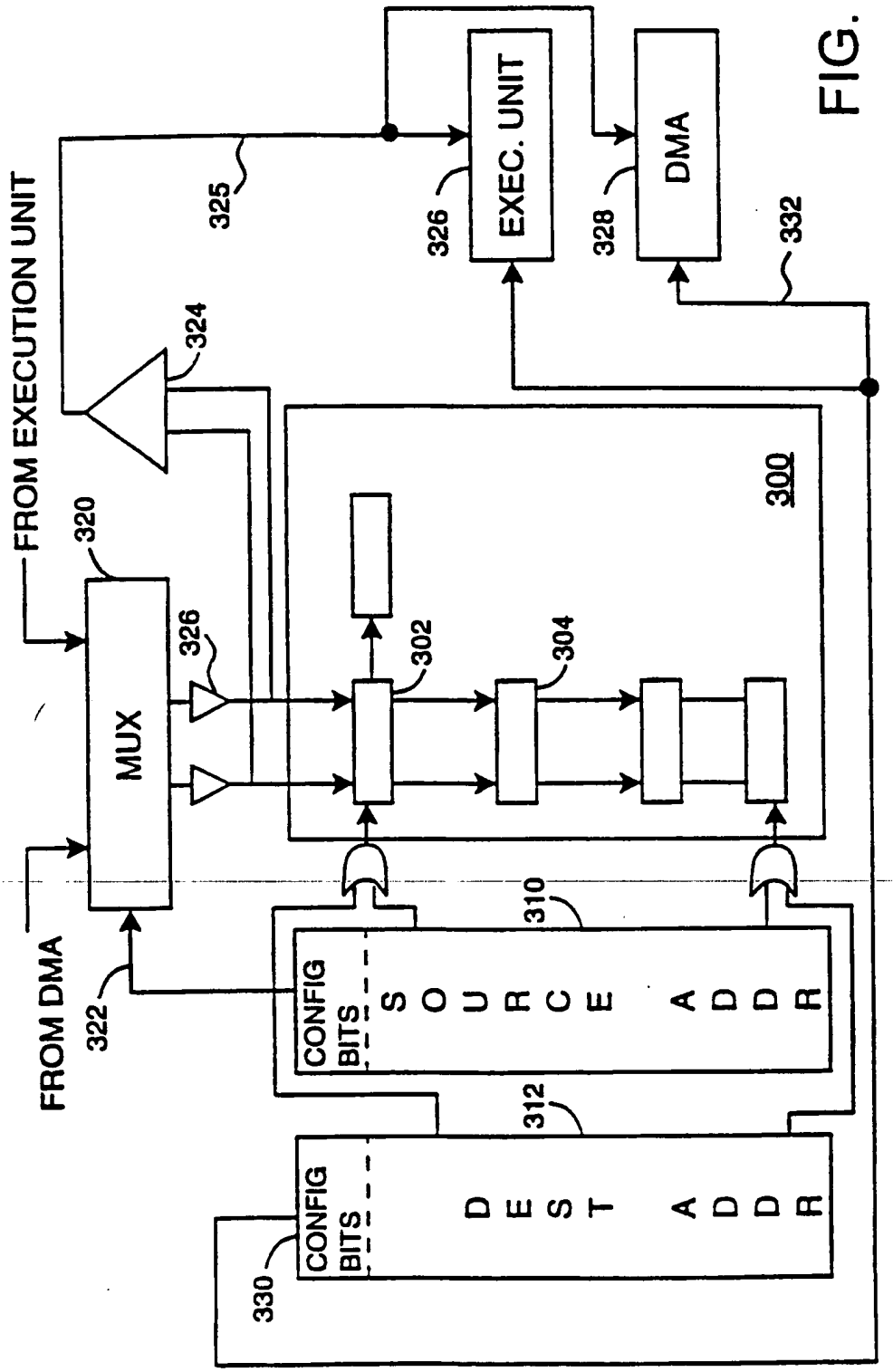

FROM EXECUTION UNIT
FROM DMA
MUX
320
322
324
325
326
328
332
EXEC. UNIT
DMA
300
302
304
310
312
330
CONFIG BITS
SOURCE ADDR
CONFIG BITS
DEST ADDR

FIG. 11A

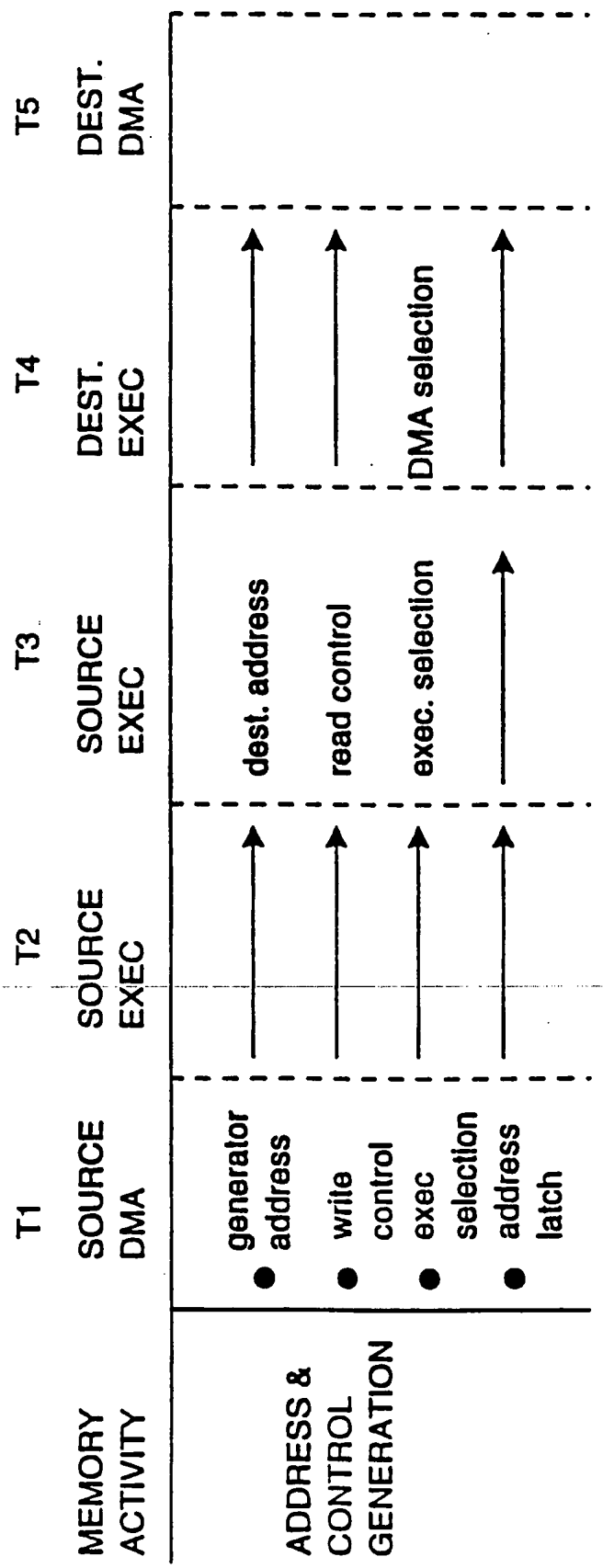

MEMORY ACTIVITY
ADDRESS & CONTROL GENERATION
T1
SOURCE DMA
T2
SOURCE EXEC
T3
SOURCE EXEC
T4
DEST. EXEC
T5
DEST. DMA
generator address
write control
exec selection
address latch
dest. address
read control
exec. selection
DMA selection

FIG. 11B

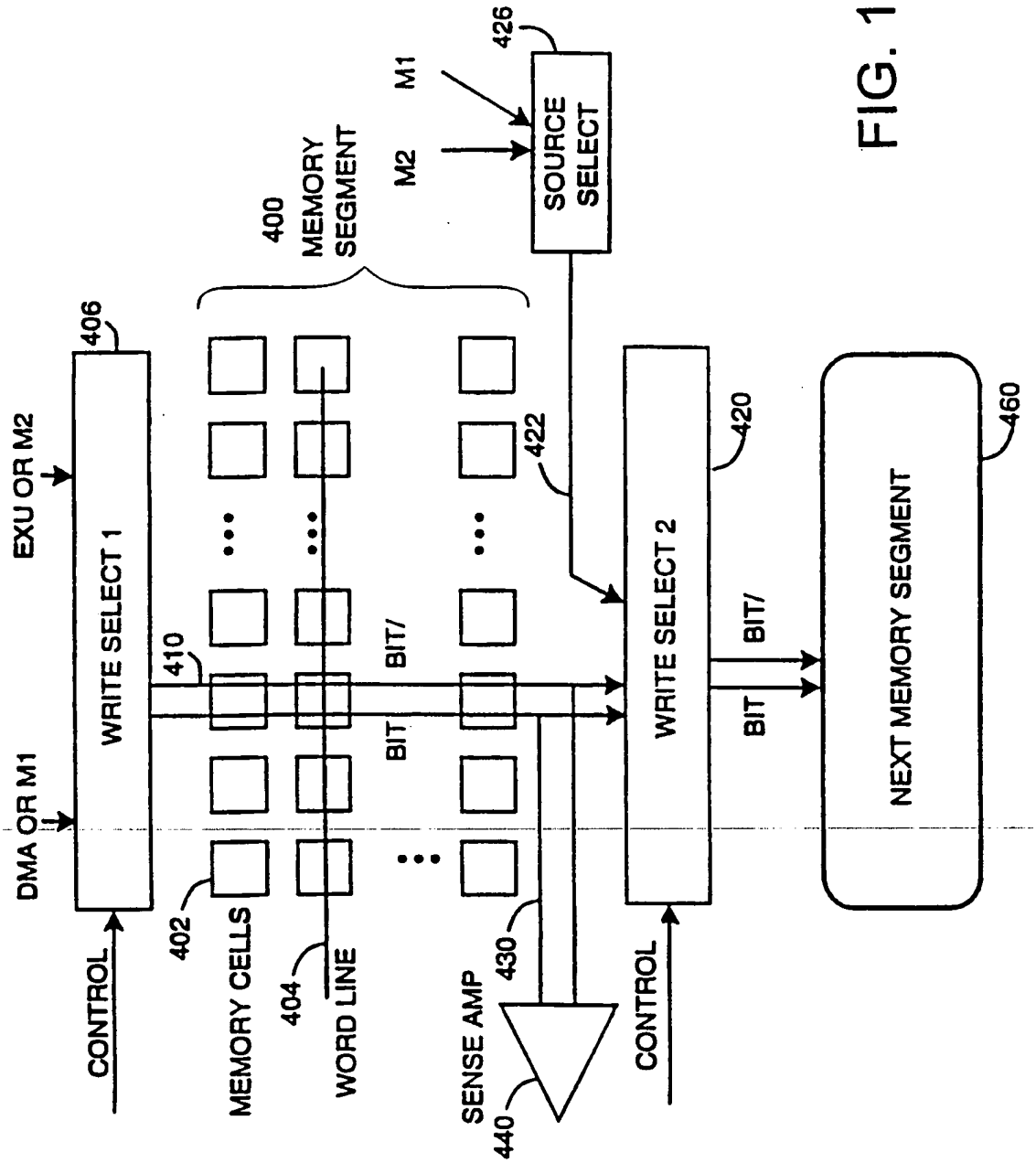

426
M1
M2
SOURCE SELECT
400
MEMORY SEGMENT
406
EXU OR M2
WRITE SELECT 1
DMA OR M1
CONTROL
410
402
MEMORY CELLS
404
WORD LINE
BIT/
BIT
422
420
WRITE SELECT 2
BIT/
BIT
460
NEXT MEMORY SEGMENT
SENSE AMP
430
440
CONTROL

FIG. 12

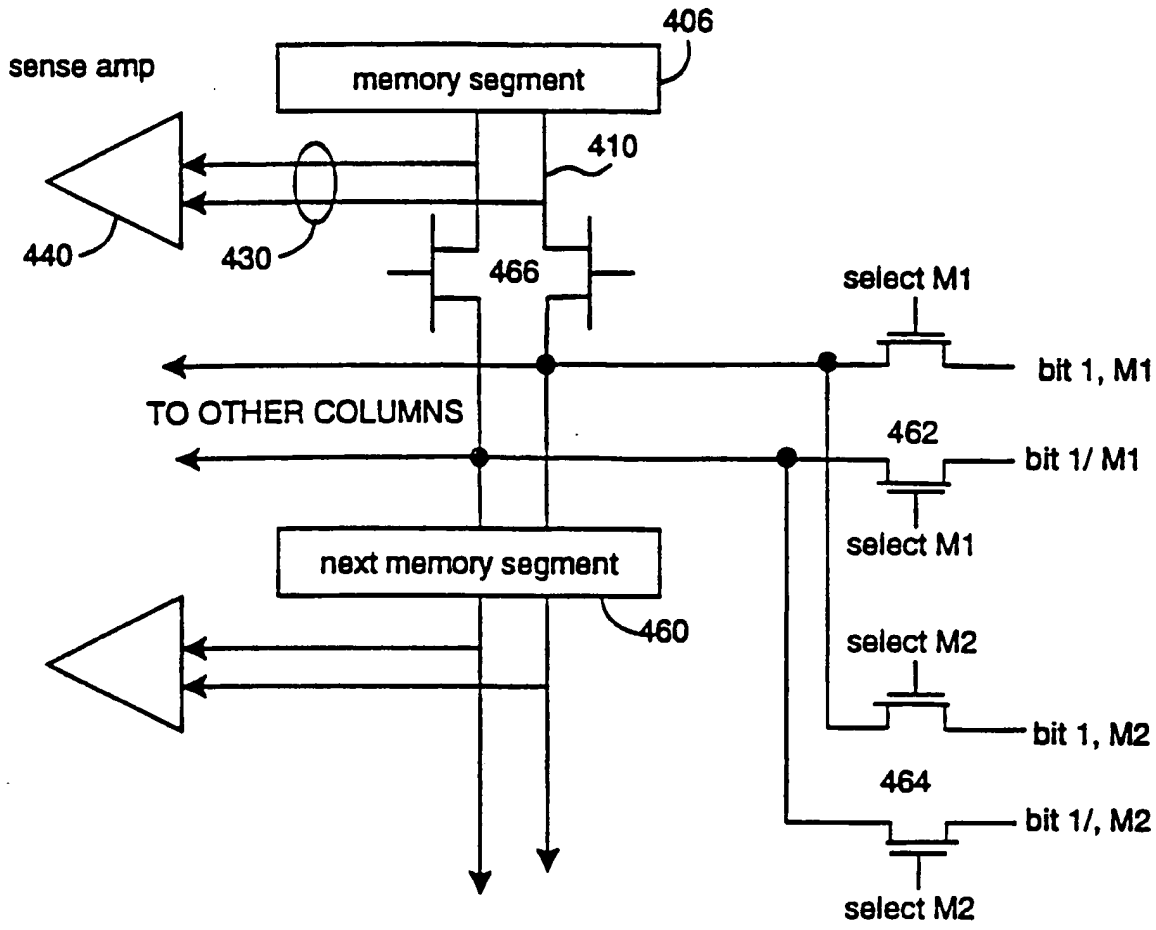


FIG. 13A

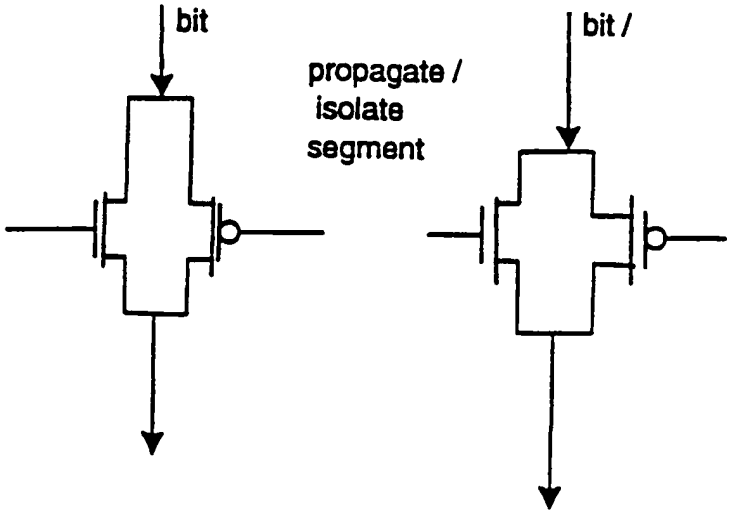


FIG. 13B

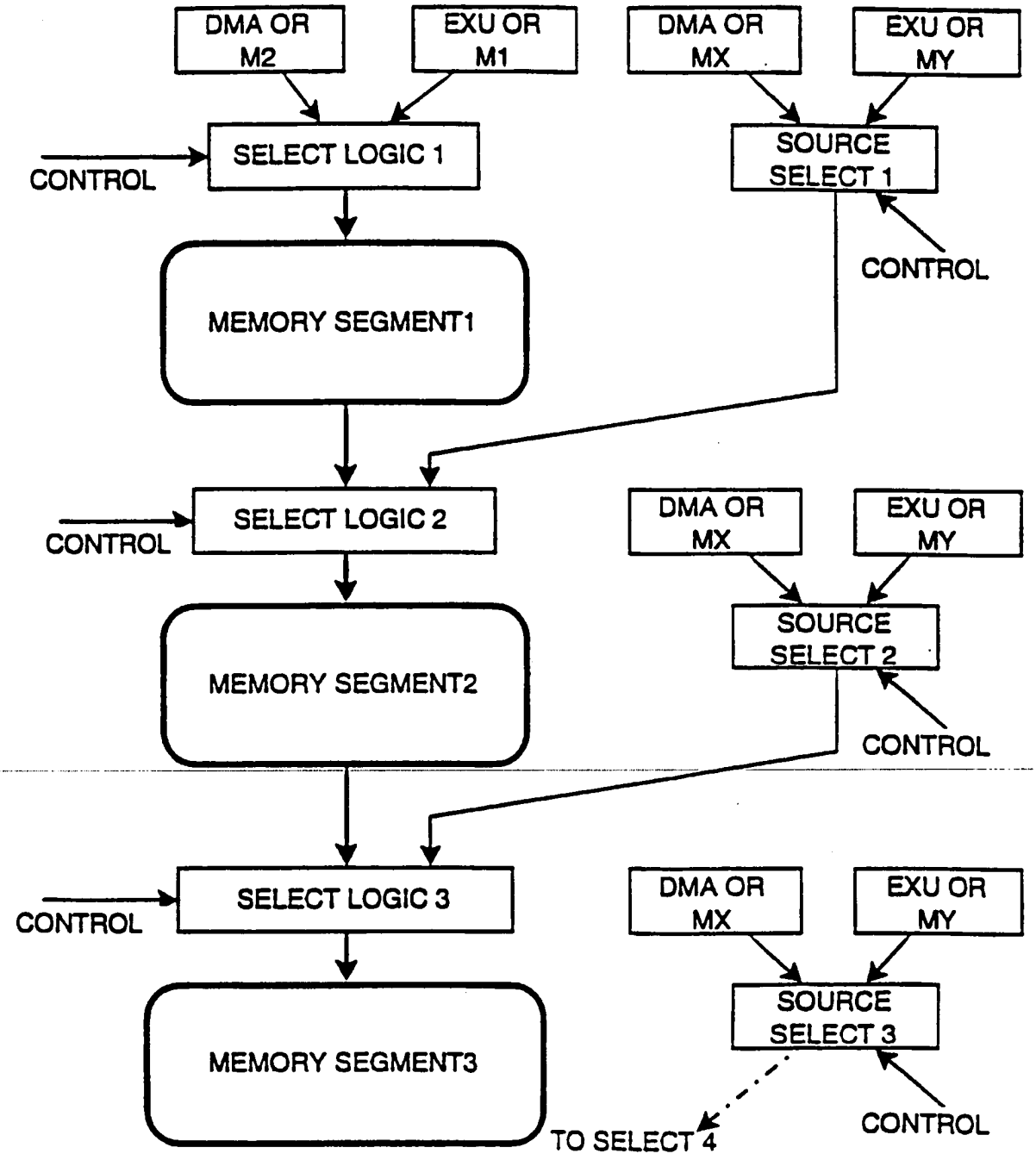

DMA OR M2
EXU OR M1
DMA OR MX
EXU OR MY
CONTROL
SELECT LOGIC 1
SOURCE SELECT 1
CONTROL
MEMORY SEGMENT1
CONTROL
SELECT LOGIC 2
DMA OR MX
EXU OR MY
SOURCE SELECT 2
MEMORY SEGMENT2
CONTROL
CONTROL
SELECT LOGIC 3
DMA OR MX
EXU OR MY
SOURCE SELECT 3
MEMORY SEGMENT3
TO SELECT 4
CONTROL

FIG. 14

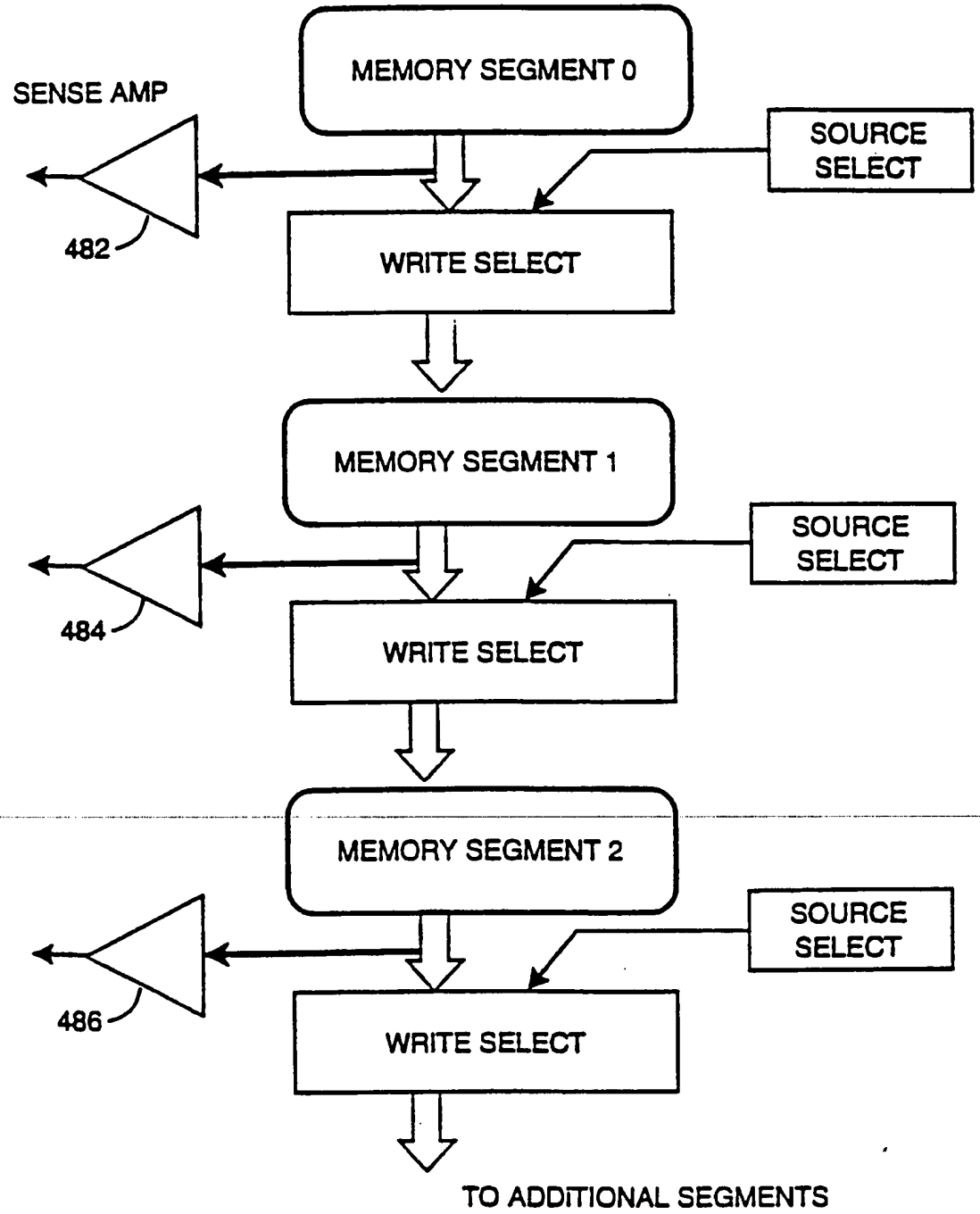

MEMORY SEGMENT 0
SENSE AMP
SOURCE SELECT
482
WRITE SELECT
MEMORY SEGMENT 1
SOURCE SELECT
484
WRITE SELECT
MEMORY SEGMENT 2
SOURCE SELECT
486
WRITE SELECT
TO ADDITIONAL SEGMENTS

FIG. 15

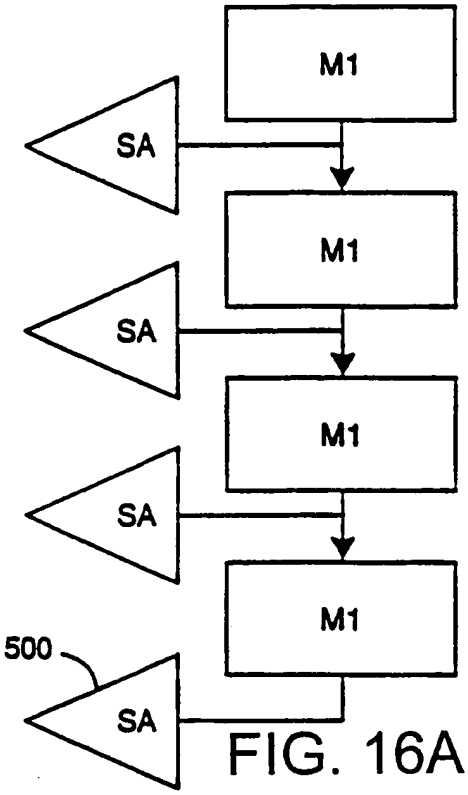


FIG. 16A

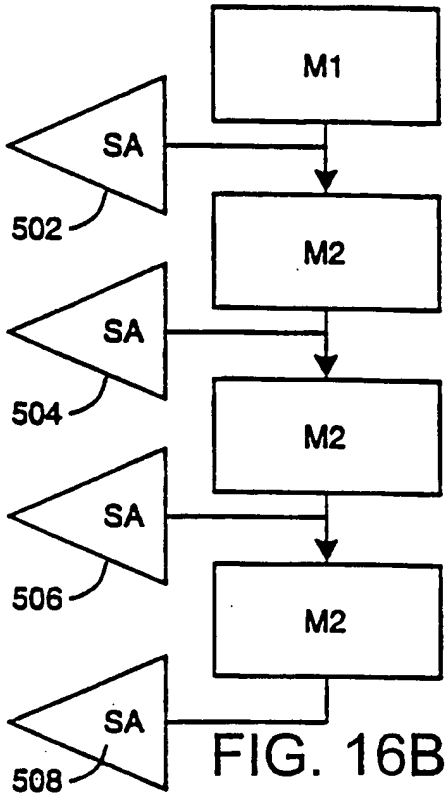


FIG. 16B

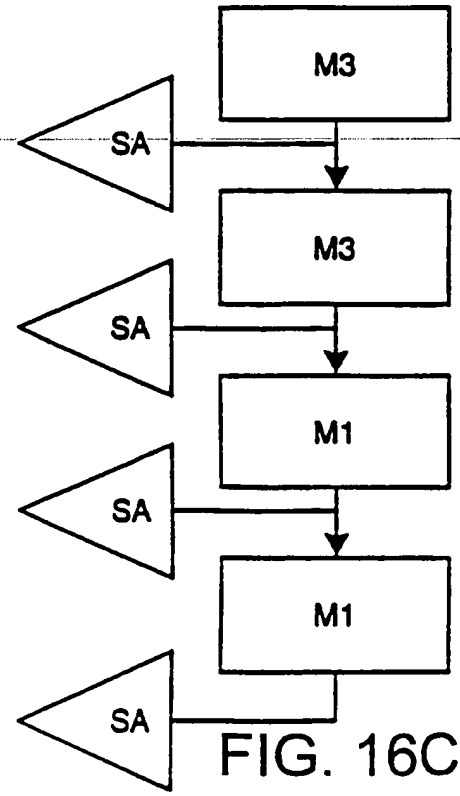


FIG. 16C

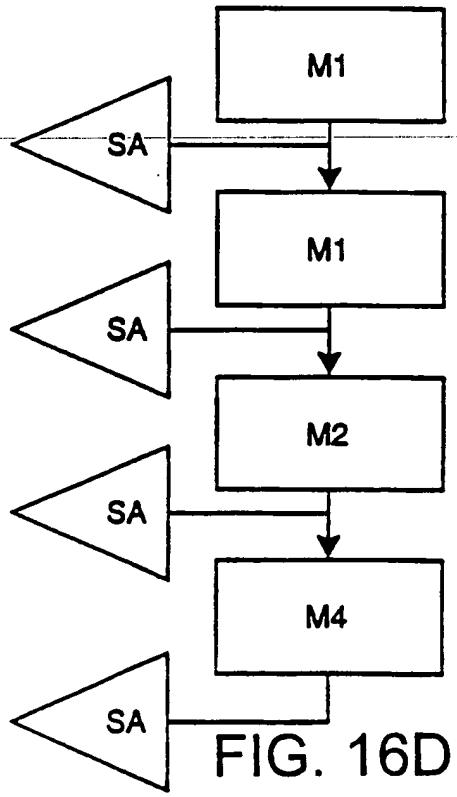


FIG. 16D

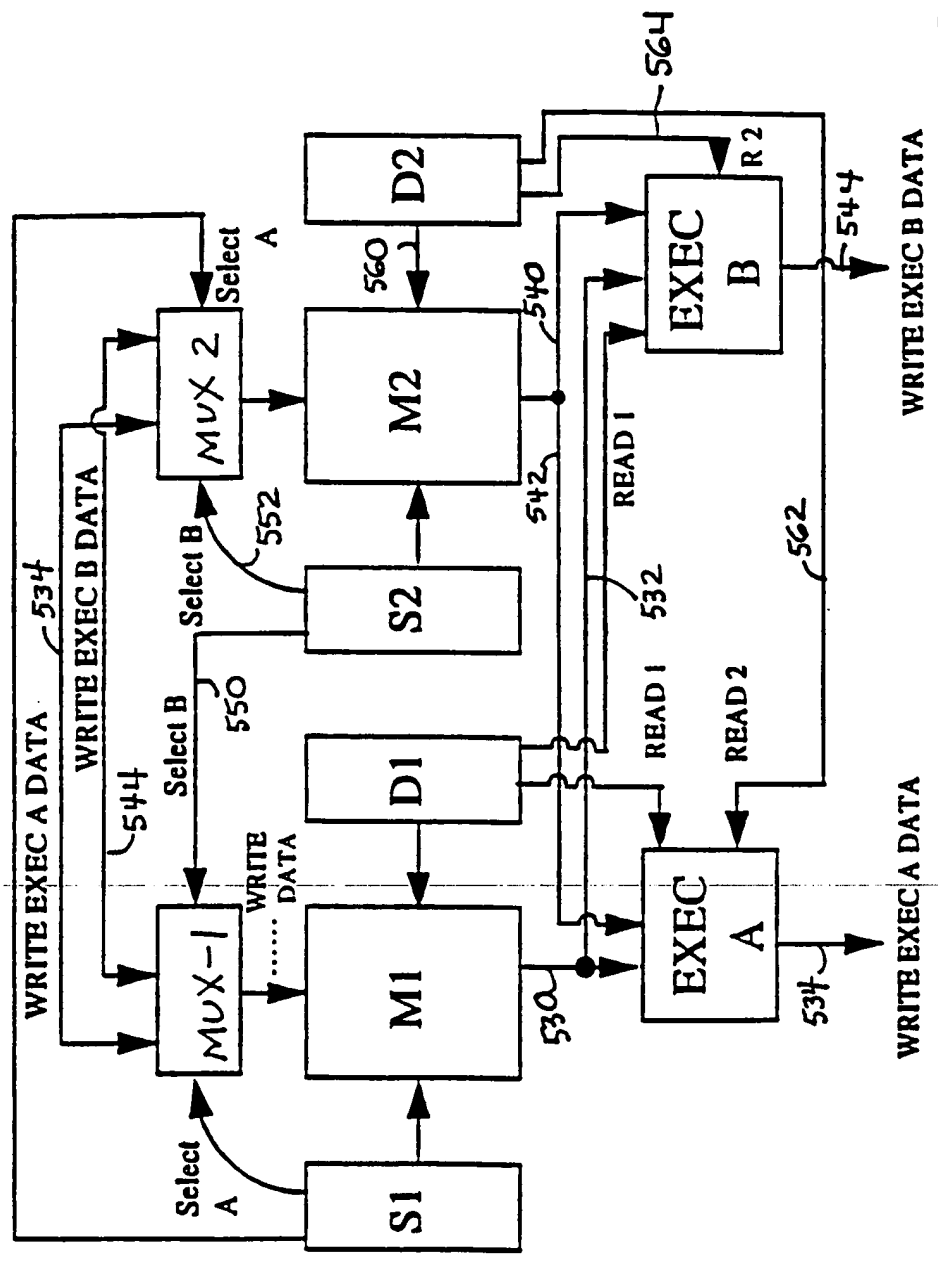

WRITE EXEC A DATA
534
WRITE EXEC B DATA
544
Select A
MUX 2
MUX-1
Select B
552
550
WRITE DATA
S1
S2
D1
D2
M1
M2
560
540
542
530
532
562
564
READ 1
READ 2
R 2
EXEC B
EXEC A
544
534
WRITE EXEC B DATA
WRITE EXEC A DATA

FIG. 17

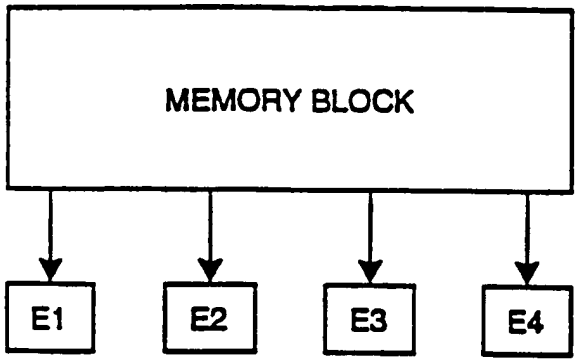


FIG. 18A

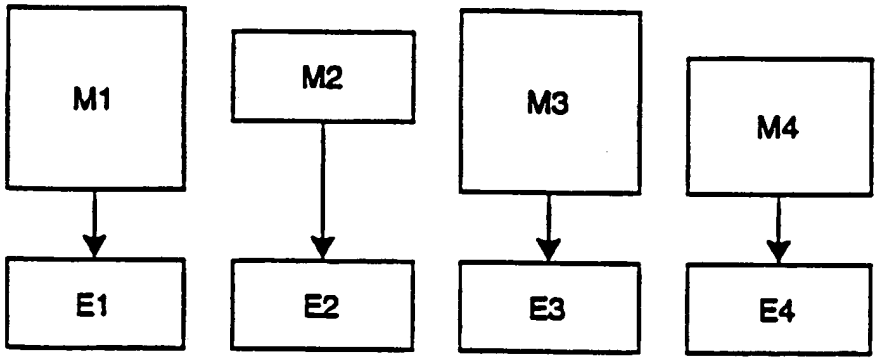


FIG. 18B

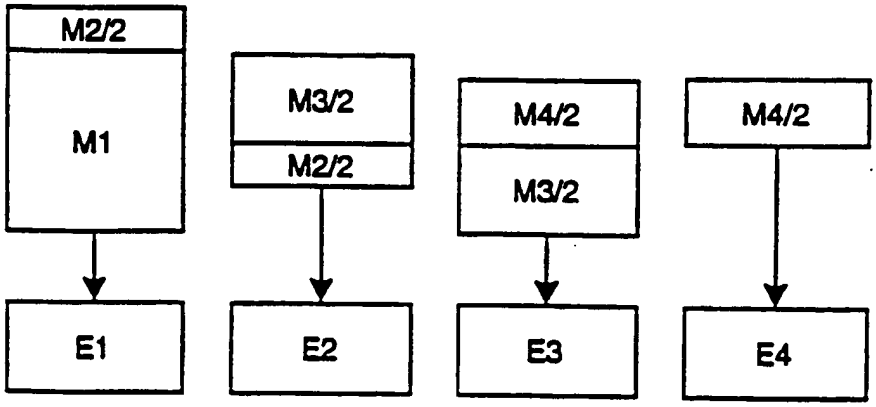


FIG. 18C

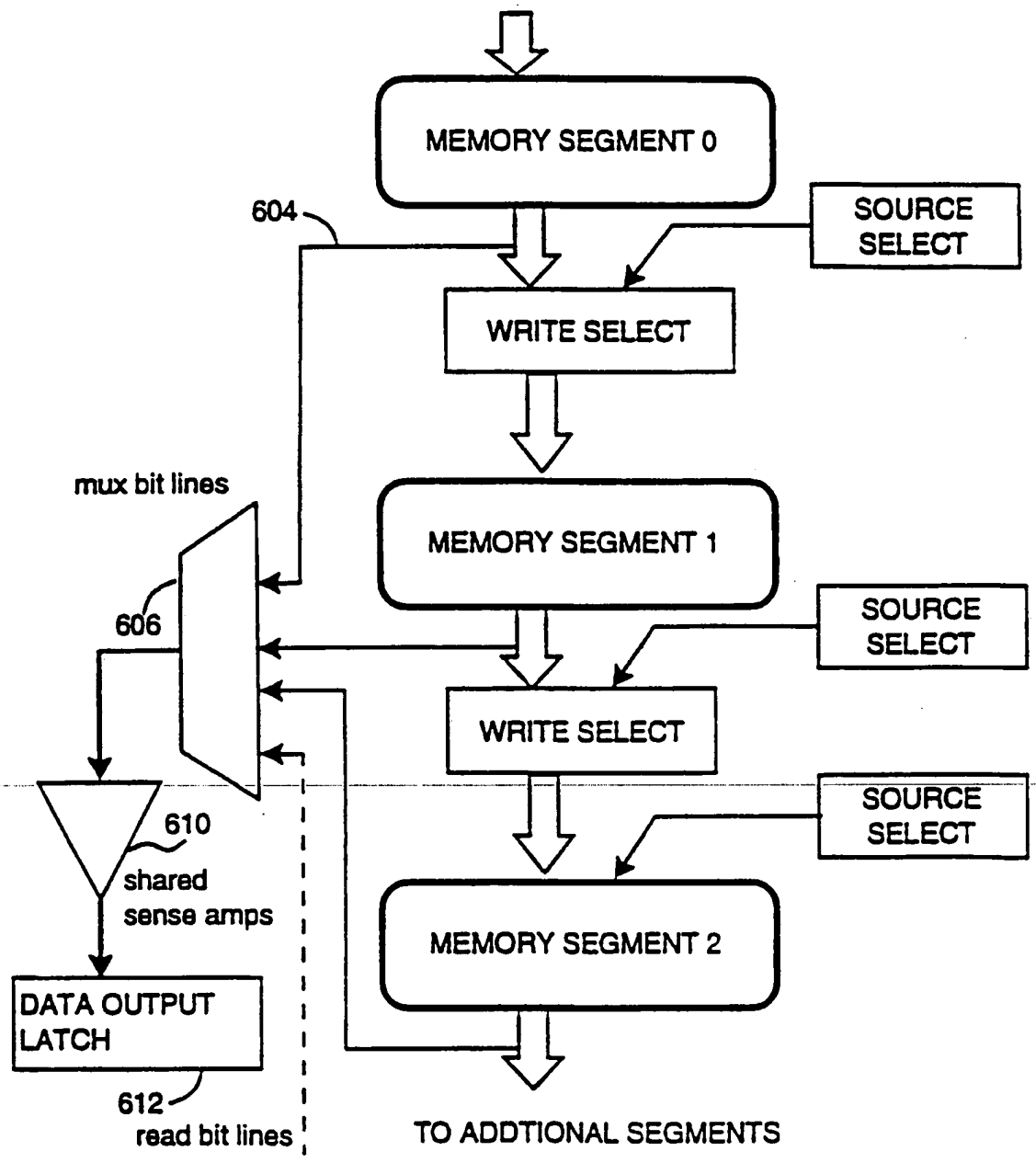

MEMORY SEGMENT 0
604
SOURCE SELECT
WRITE SELECT
mux bit lines
MEMORY SEGMENT 1
606
SOURCE SELECT
WRITE SELECT
SOURCE SELECT
610
shared sense amps
MEMORY SEGMENT 2
DATA OUTPUT LATCH
612
read bit lines
TO ADDTIONAL SEGMENTS

FIG. 19

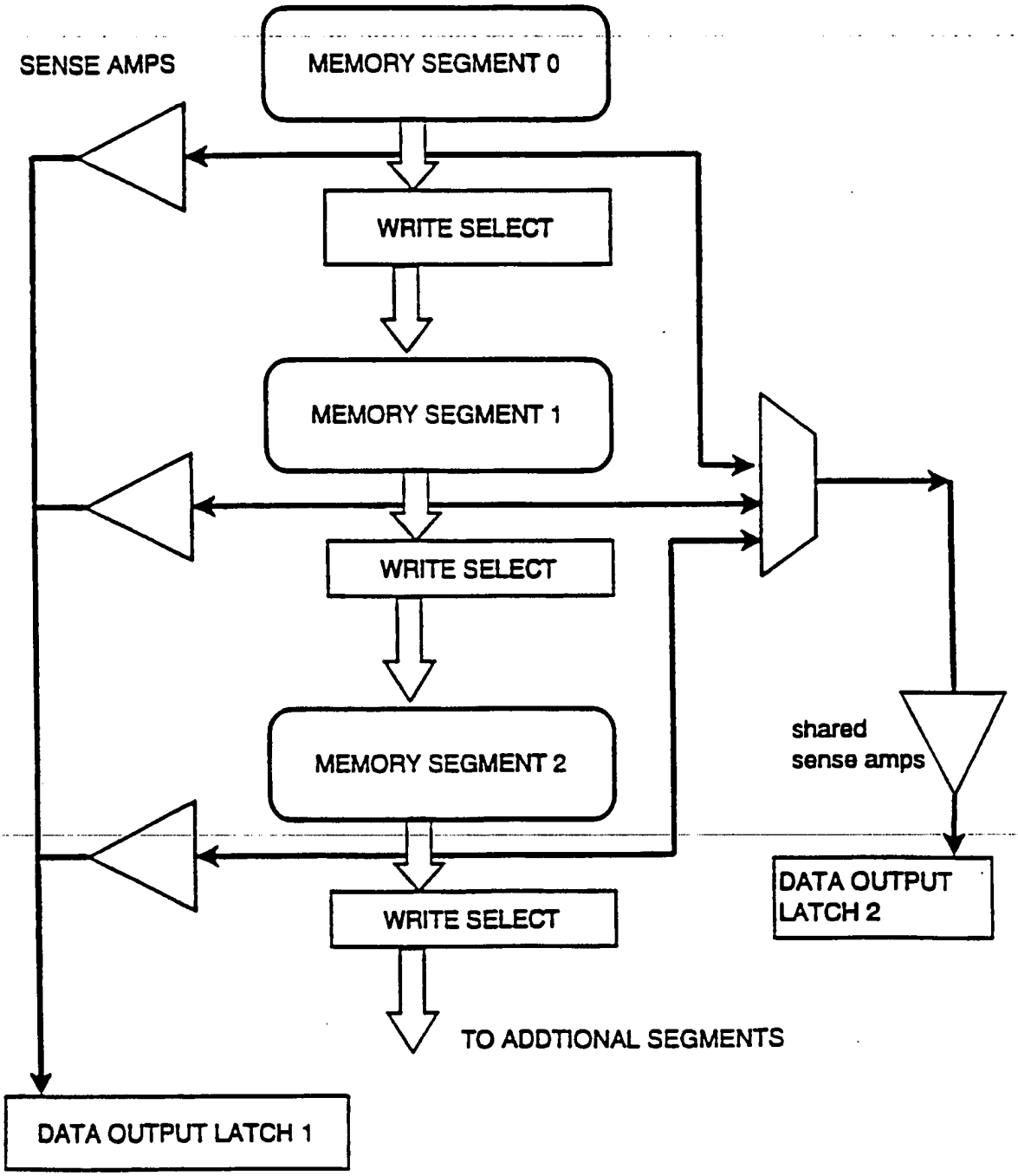

SENSE AMPS
MEMORY SEGMENT 0
WRITE SELECT
MEMORY SEGMENT 1
WRITE SELECT
MEMORY SEGMENT 2
WRITE SELECT
shared
sense amps
DATA OUTPUT
LATCH 2
TO ADDTIONAL SEGMENTS
DATA OUTPUT LATCH 1

FIG. 20

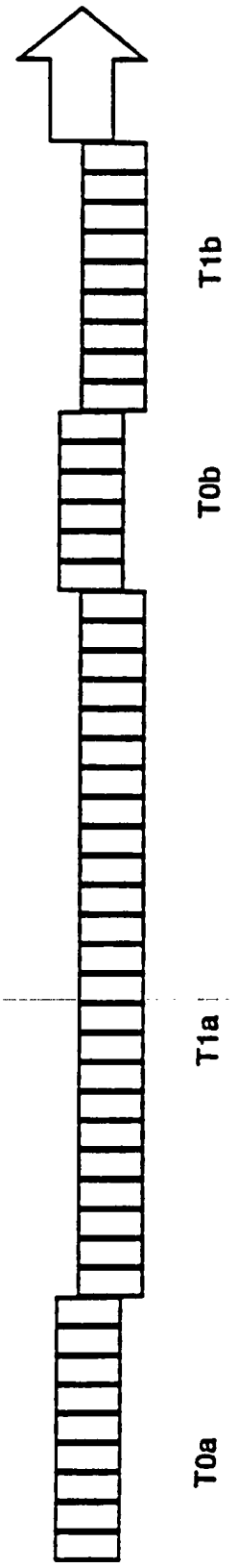


FIG. 21

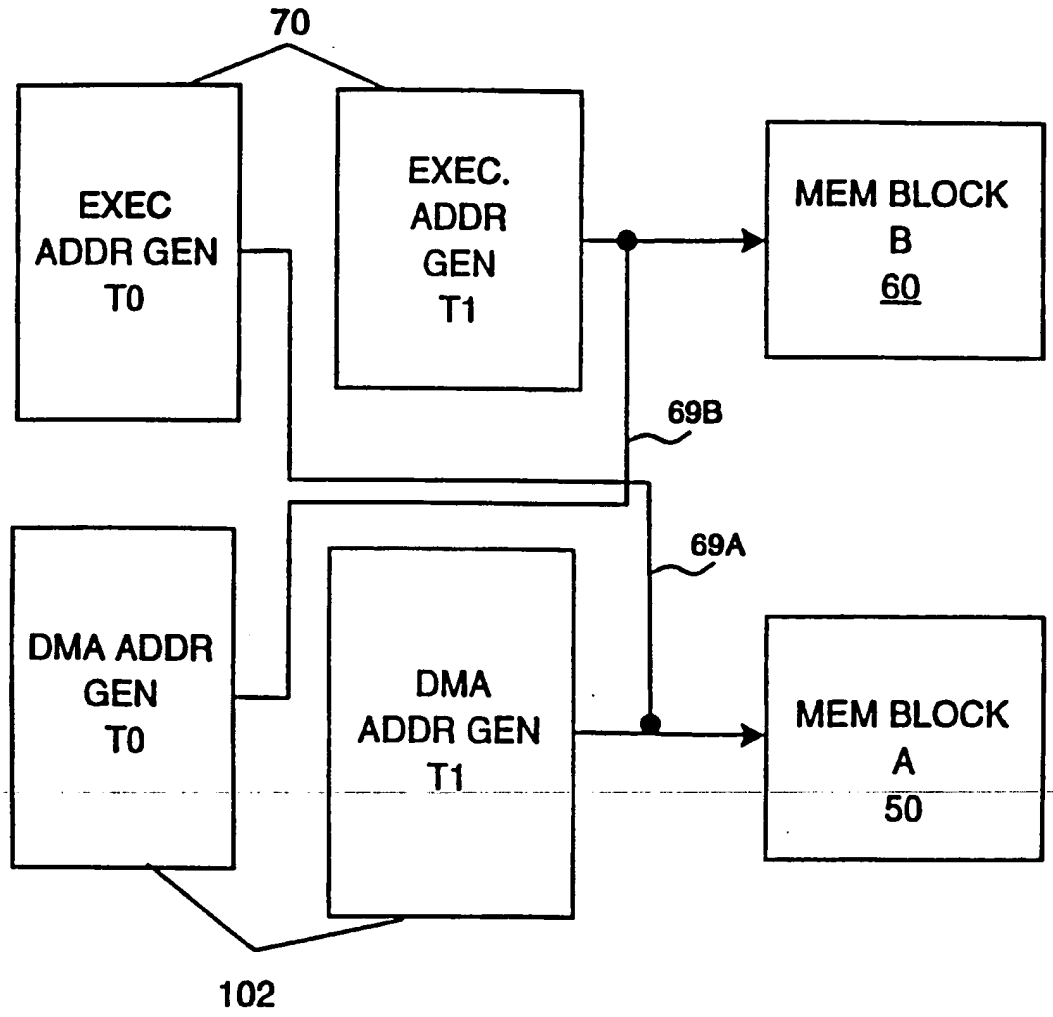

70
EXEC ADDR GEN T0
EXEC. ADDR GEN T1
MEM BLOCK B 60
69B
69A
DMA ADDR GEN T0
DMA ADDR GEN T1
MEM BLOCK A 50
102

FIG. 22A

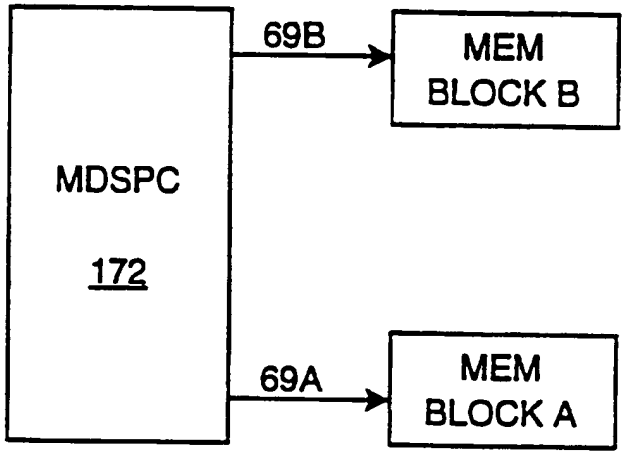


FIG. 22B

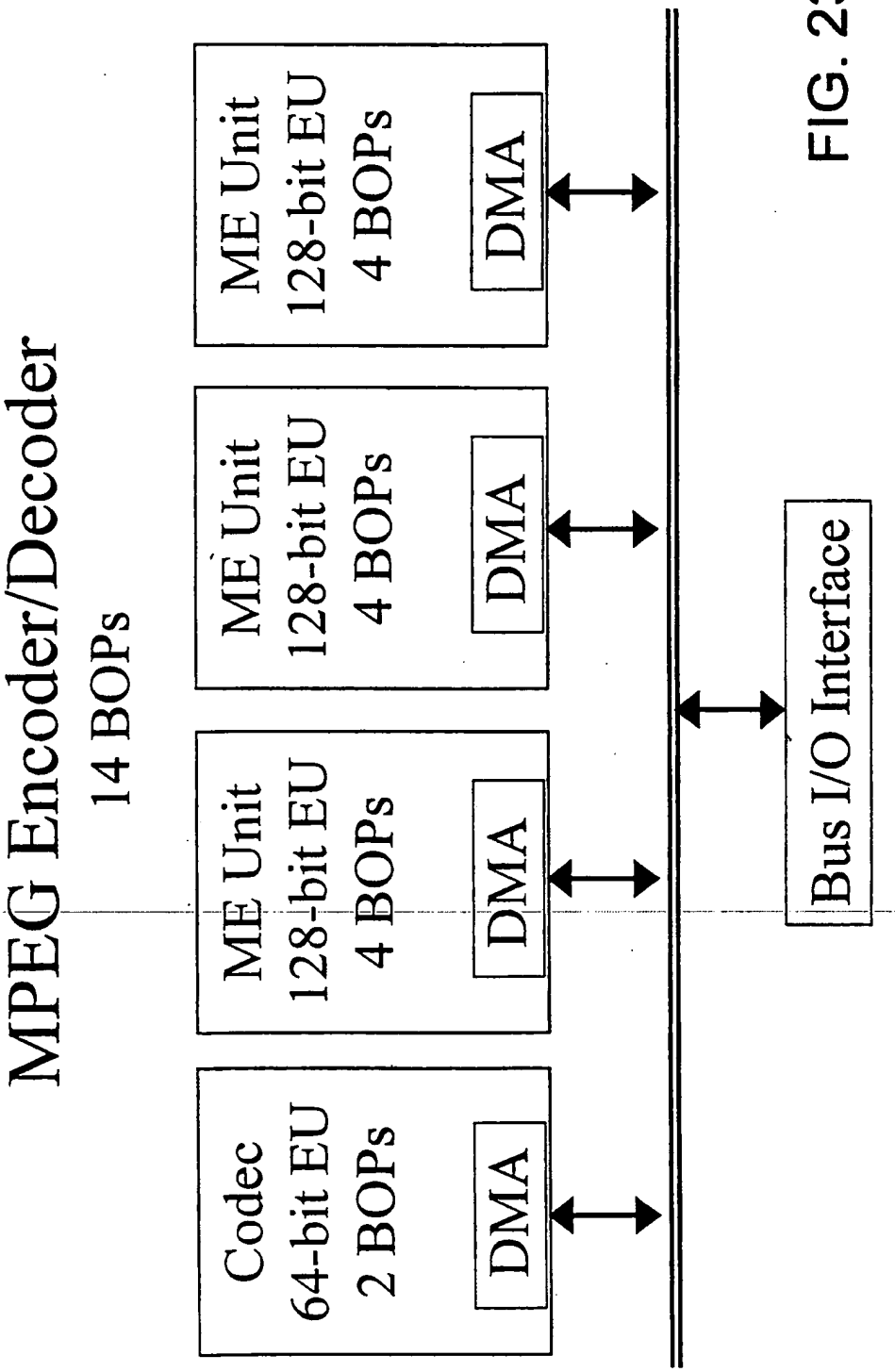

MPEG Encoder/Decoder
14 BOPs
Codec
64-bit EU
2 BOPs
DMA
ME Unit
128-bit EU
4 BOPs
DMA
ME Unit
128-bit EU
4 BOPs
DMA
ME Unit
128-bit EU
4 BOPs
DMA
Bus I/O Interface

FIG. 23

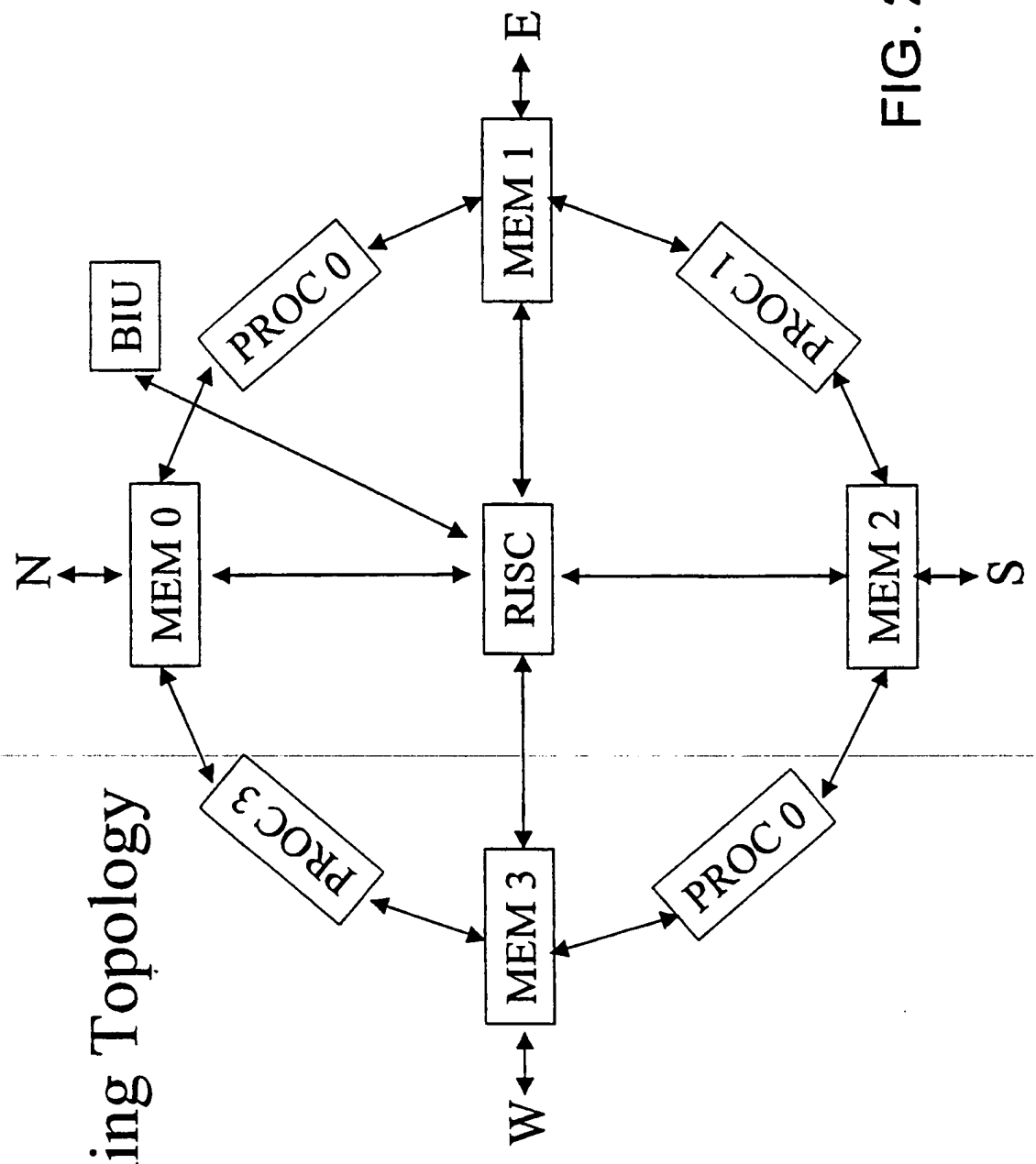

Ring Topology
N
S
E
W
MEM 0
MEM 1
MEM 2
MEM 3
RISC
BIU
PROC 0
PROC 1
PROC 2
PROC 3

FIG. 24

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/07771

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) : G06F 15/76
US CL :345/519

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 345/519, 505, 506, 520; 709/251; 712/10, 14, 15, 20, 21, 23, 41; 370/258; 375/212.

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

NONE

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS AND DIALOG: Search terms: ring, risc processor, plural? processor, graphics, memory block or plural? memory

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y, P	US 5,841,444 A (MUN ET AL) 24 NOVEMBER 1998, FIGURE 2, AND COLUMNS 3 AND 4	1-13
Y	US 5,659,543 A (ATER ET AL) 19 AUGUST 1997, ABSTRACT, FIGURES 1A AND 1B.	1-13
Y	GUTTAG ET AL, A SINGLE CHIP MULTIPROCESSOR FOR MULTIMEDIA: THE MVP, IEEE COMPUTER GRAPHICS & APPLICATIONS, NOVEMBER 1992, FIGURE 4, PAGES 60-64.	1-13
A	FUJITA ET AL, A DATAFLOW IMAGE PROCESSING SYSTEM TIP-4, PROCEEDINGS OF THE 5TH INTERNATIONAL CONFERENCE ON IMAGE ANALYSIS AND PROCESSING, 20-22 SEPTEMBER, 1989, PAGES 735-740.	1-13

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 26 MAY 1999	Date of mailing of the international search report 11 JUN 1999
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer KEE M. TUNG Telephone No. (703) 305-9660

Form PCT/ISA/210 (second sheet)(July 1992) ★